RÉPUBLIQUE FRANÇAISE
Liberté Égalité Fraternité

Université de Toulouse

Doctorat de l'Université de Toulouse

préparé à l'ISAE-SUPAERO

Vérification formelle des systèmes à haute dimension fondés sur l'apprentissage machine (ML)

Thèse présentée et soutenue, le 7 juillet 2026 par

Noémie COHEN

École doctorale
EDMITT - Ecole Doctorale Mathématiques, Informatique et Télécommunications de Toulouse

Spécialité
Informatique et Télécommunications

Unité de recherche
ISAE-ONERA MOIS MOdélisation et Ingénierie des Systèmes

Thèse dirigée par
Claire PAGETTI, Xavier PUCEL et Mélanie DUCOFFE

Composition du jury
M. Pierre-Loïc GAROCHE, Président, ENAC
M. Benedikt BOLLIG, Rapporteur, CNRS/LMF
M. Matthieu MARTEL, Rapporteur, Université de Perpignan Via Domitia
M. Antoine MINÉ, Examinateur, Sorbonne Université
Mme Caterina URBAN, Examinatrice, INRIA
M. Arthur CLAVIÈRE, Examinateur, Collins Aerospace
Mme Claire PAGETTI, Directrice de thèse, ONERA
Mme Mélanie DUCOFFE, Co-directrice de thèse du monde socio-économique, Airbus Operations S.A.S

Membres invités
M. Christophe GABREAU, Airbus Operations S.A.S
M. Xavier PUCEL, Co-directeur de thèse, ONERA

Remerciements

Ce travail n'aurait pas été possible sans le soutien d'Airbus et de l'Association nationale de la recherche et de la technologie (ANRT), qui ont financé cette thèse CIFRE. Je remercie également ANITI (Institut toulousain de l'intelligence artificielle et naturelle), notamment le projet DEEL et la chaire de recherche sur l'embarquabilité, pour leur environnement scientifique stimulant.

Je tiens tout d'abord à remercier les membres de mon jury, Benedikt Bollig, Arthur Clavière, Pierre-Loïc Garoche, Matthieu Martel, Antoine Miné et Caterina Urban. Merci pour votre lecture attentive, la richesse de nos échanges et les nombreuses pistes ouvertes pour la poursuite de ces travaux.

Je tiens à exprimer toute ma gratitude à mes encadrants. Claire Pagetti, ma directrice de thèse, merci pour ton engagement et ton expertise tout au long de ces trois années. Ta disponibilité et ta générosité ont été constantes. Merci également pour ton exigence, qui m'a toujours poussée à produire un travail de qualité.

Mélanie Ducoffe, c'est lors de l'un de tes cours que j'ai découvert ce sujet de thèse. Merci pour ton expertise et ton enthousiasme communicatif pour cette thématique. Ton regard scientifique ont continué à nourrir mes réflexions tout au long de ces travaux.

Xavier Pucel, merci pour ton expertise et ta pédagogie. Tes intuitions et tes schémas éclairants ont souvent permis de clarifier des notions complexes. Nos échanges étaient toujours inspirants et bienveillants.

Christophe Gabreau, merci de m'avoir transmis de ta connaissance du milieu aéronautique et de m'avoir ouvert les portes du vaste monde de la certification. Tu as toujours su replacer mes travaux dans leur contexte, en rappeler les enjeux et, surtout, leur utilité.

J'ai par ailleurs eu la chance d'encadrer Timothée Fayard lors de son stage.

Merci à tous les collègues et amis de l'ONERA et d'Airbus. Merci à Iryna et Damien pour leur accueil attentionné. Merci à Cédric, qui a su me faire éclater de rire jusque dans les moments les moins drôles et qui n'a jamais manqué de venir me saluer chaque matin. Merci à Anthony, mon frère de thèse et compagnon de bureau : pas besoin de parler, nous nous comprenons. Merci aux membres de la Paget'team. Merci Baptiste, Janelle, Sofiane et à toute la compagnie pour ces sorties revigorantes. Comme le dit ma mère, le plus important, ce sont les rencontres ; je suis heureuse que nos chemins se soient croisés.

Merci à mes amis, qui ont vu ma maison en sucre devenir une maison en dur et qui ont toujours su trouver le mot pour rire face aux subtilités de la recherche ou de mon sujet. Merci notamment à Camille, Marie et Emma pour votre amitié fidèle.

Je remercie mes parents ainsi que Léa et Nathan pour leur soutien constant et leurs conseils. Cette thèse aura aussi été marquée par la naissance d'Andréa et d'Alix, qui n'ont pas contribué au manuscrit, mais à mon plus grand bonheur.

Prakkas, tu as été présent depuis la toute première évocation du mot « thèse » jusqu'au jour de la soutenance, où tu as joué le rôle décisif de technicien en visioconférence. Pour ta fidélité, ton écoute, ton sens de l'anticipation et ton soutien, merci.

Abstract

Context: The integration of machine learning into safety-critical avionic systems requires a rigorous level of verification to fit the certification requirements. The problem is that traditional software development relies on requirement-based, while machine learning disrupts this process with data-driven techniques. Thus, new activities have to be introduced in the development process. Among them, formal verification is a means of compliance to guarantee the robustness of a neural network against input perturbations.

Example: A relevant application is the vision-based landing system, such as the Autonomous Taxi, Take-Off, and Landing (ATTOL) project. In this system, an object detection neural network processes high-dimensional pictures to locate the airport runway. A failure to correctly detect the runway under different lighting conditions or camera rotations could compromise the safety of the aircraft.

Assumptions: We restrict our study to the inference phase of sequential feed-forward neural networks using ReLU activations. We investigate two specific types of visual variations applied to a reference picture: geometric transformations (i.e rotation) and pixel-intensity perturbations (such as white-noise, brightness, and contrast).

Contributions: To verify the robustness of vision-based neural networks, we introduce two main methodologies. First, we present Super-DeepG, a fast approach to verify robustness against geometric transformations. We replace computationally heavy linear programming solvers with analytical tensor operations. This reformulation allows us to evaluate thousands of pixels simultaneously on a GPU, bridging the gap between mathematical precision and computational scalability. Second, we extend formal verification beyond simple classification to the object detection task. We mathematically prove that the Intersection over Union (IoU) metric is unimodal with respect to each bounding box coordinate. Based on this property, we introduce the Unimodal-IoU method. This technique allows us to verify the robustness of bounding box predictions under visual perturbations.

Résumé en français

Contexte : L'intégration de l'apprentissage automatique dans les systèmes critiques de l'avionique exige une certification rigoureuse. Le problème réside dans le fait que le développement logiciel traditionnel repose sur une approche fondée sur les exigences, tandis que l'apprentissage automatique bouleverse ce processus en introduisant des techniques fondées sur les données. De nouvelles activités doivent donc être intégrées au processus de développement. Parmi celles-ci, la vérification formelle constitue un moyen de conformité permettant de garantir la robustesse d'un réseau de neurones face aux perturbations des données d'entrée.

Hypothèses : Nous restreignons notre étude à la phase d'inférence des réseaux de neurones séquentiels utilisant des fonctions d'activation ReLU. Nous étudions deux types spécifiques de variations visuelles appliquées à une image de référence : les transformations géométriques (comme les rotations par exemple) et les perturbations de l'intensité des pixels (comme le bruit blanc, la luminosité et le contraste).

Exemple : Une application pertinente est le système d'atterrissage basé sur la vision, tel que le projet *ATTOL (Autonomous Taxi, Take-Off, and Landing)*. Dans ce système, un réseau de neurones de détection d'objets analyse des images en haute dimension pour localiser la piste d'atterrissage. Une incapacité à détecter correctement la piste sous différentes conditions d'éclairage ou inclinaisons de la caméra pourrait compromettre la sécurité de l'avion.

Contributions : Afin de vérifier la robustesse des réseaux de neurones pour des tâches en haute dimension, nous présentons deux méthodologies principales. Premièrement, nous proposons Super-DeepG, une approche pour certifier la robustesse face aux transformations géométriques. Nous remplaçons les solveurs de programmation linéaire, très lourds en calcul, par des opérations analytiques basées sur des tenseurs Pytorch. Cette reformulation nous permet d'évaluer des milliers de pixels de manière simultanée sur un GPU, alliant ainsi précision mathématique et passage à l'échelle. Deuxièmement, nous étendons la vérification formelle au-delà de la simple classification pour aborder la détection d'objets. Nous prouvons mathématiquement que la métrique d'Intersection sur Union (IoU) est unimodale pour chaque coordonnée de la boîte englobante. En nous appuyant sur cette propriété, nous introduisons la méthode Unimodal-IoU. Cette technique permet de vérifier formellement l'exactitude des prédictions des boîtes englobantes soumises à des perturbations visuelles.

Contents

List of Figures **9**

List of Tables **9**

List of Acronyms **9**

List of Symbols **9**

List of Publications **10**

1 Introduction **11**
1.1 General safety concepts . . . 12
1.2 Impact of ML in the certification method . . . 14
1.3 Robustness verification of neural networks . . . 16
1.4 Use case . . . 18
1.5 Adopted approach . . . 20
1.5.1 Choice of perturbations for the robustness property . . . 20
1.5.2 Methodology and main results . . . 21
1.6 Organization of the Manuscript . . . 22

I Background **24**

2 Background **26**
2.1 Verification of reachability properties . . . 26
2.1.1 Reachability property . . . 26
2.1.2 Input domain approximation . . . 27
2.1.3 Computation of the (reachable) image of $\mathcal{C}$ under f . . . 29
2.1.4 Trade-offs in over-approximations . . . 30
2.2 Sequential feed-forward neural networks . . . 31
2.2.1 Definition . . . 31
2.2.2 Example of a reachability property on classification . . . 32
2.3 LiRPA: Neural networks' linear relaxation . . . 34
2.3.1 Definition of linear relaxation . . . 34
2.3.2 Computing the input domain of a layer . . . 35
2.3.3 Computing the linear relaxation coefficients per layer . . . 37
2.3.4 Composition of linear relaxations . . . 38
2.3.5 Special case: constant relaxation IBP . . . 40
2.3.6 The CROWN method . . . 41
2.4 Auto-LiRPA . . . 44
2.4.1 Pytorch basics . . . 44

2.4.2 `auto-LiRPA` . . . 45
2.4.3 Example: PlusConstant custom function . . . 47
2.4.4 More complex input domains . . . 50
2.5 Conclusion . . . 51

II Contributions **52**

3 Certifying geometric robustness of neural networks **54**
3.1 Geometric transformation . . . 54
3.1.1 Picture . . . 55
3.1.2 Coordinate transformation . . . 55
3.1.3 Picture transformation . . . 57
3.1.4 General formulation of geometric transformations . . . 60
3.1.5 Implementation of geometric transformations in PyTorch . . . 61
3.2 Geometric robustness: Problem statement . . . 62
3.2.1 Characterising the set of transformed pictures . . . 62
3.2.2 Relaxation of the set of transformed pictures . . . 64
3.3 Formal methods for geometric robustness . . . 67
3.3.1 Strategy 1: Operator-wise Bounding (IBP-based) . . . 68
3.3.2 Strategy 2: Lipschitz optimization . . . 74
3.3.3 Strategy steps . . . 77
3.4 Comparison of Operator-wise and Lipschitz-based Strategies . . . 80

4 Overview of Super-DeepG approach **82**
4.1 STEP 1: Unsound linear bounds . . . 83
4.1.1 STEP 1.A: Problem statement . . . 83
4.1.2 STEP 1.B: DeepG and PWL resolution . . . 87
4.1.3 STEP 1.C: Super-DeepG resolution . . . 89
4.1.4 STEP 1: Conclusion . . . 100
4.2 STEP 2: Lipschitz constant . . . 100
4.2.1 STEP 2.A: Problem statement . . . 101
4.2.2 STEP 2.B and STEP 2.C DeepG, PWL and Super-DeepG resolution . . . 105
4.3 STEP 3: Sound linear bounds . . . 105
4.3.1 STEP 3A: Problem statement . . . 105
4.3.2 STEP 3.B: DeepG and PWL resolution . . . 110
4.3.3 STEP 3.C: Super-DeepG resolution - multi-dimensions . . . 112
4.3.4 STEP 3: Conclusion . . . 114
4.4 Experimental evaluation . . . 115
4.4.1 Implementation . . . 115
4.4.2 Metrics . . . 115
4.4.3 Experimental Setup . . . 115
4.4.4 Comparison methodology with state of the art . . . 116
4.4.5 Result 1: Super-DeepG is faster than DeepG, with similar (or better) accuracy. . . . 117
4.4.6 Result 2: On CGT networks we match FGV's certified accuracy. 118
4.4.7 Result 4: Super-DeepG is faster than PWL while maintaining competitive precision. . . . 119

4.4.8 Result 5: Super-DeepG achieves significant geometric certification results on TinyImageNet without training specifically for geometric robustness. . . . 120
4.4.9 Influence of the hyper-parameters P and interval size. . . . 121
4.4.10 Influence of the hyper-parameters N and interval size 121
4.5 Conclusion 122

5 Robustness of Object Detection 123
5.1 Object detection 124
5.2 Problem statement 125
5.2.1 Relaxation of the object detection neural network 126
5.2.2 Robustness verification 127
5.3 Verification of object detection neural networks 129
5.3.1 Strategies for robustness verification 129
5.3.2 Our method: Bounding $\mathrm{IoU}_{b_{gt}}(b)$ using its unimodality in each coordinate 136
5.3.3 Algorithmic implementation and code 137
5.4 Experiment 138
5.4.1 Experimental Setup 138
5.4.2 Metrics and perturbations 139
5.4.3 Results 140
5.5 Related Work 141
5.5.1 Raviv et al. 142
5.5.2 Brückner et al. 145
5.6 Conclusion 148

III Conclusion and Future Work 150

6 Conclusion 152
6.1 The challenge of high-dimensional robustness 152
6.2 Robustness of object detection neural network 153
6.3 Contributions 154
6.3.1 Verifying geometric robustness of neural networks 154
6.3.2 The Super-DeepG approach 155
6.3.3 Robustness of Object Detection to Perturbations 155
6.4 Insights 156
6.5 Perspectives 157
6.5.1 Combining geometric perturbations and object detection 157
6.5.2 Open challenges for real-world perturbations 158
6.5.3 Adapting to alternative overlap metrics 1
6.6 Concluding remarks 1

A Supplementary Materials 2
A.1 Deepg algorithm outline step 3 2

List of Figures

1.1 From traditional requirement-based to neural network 11
1.2 Certification specification and related standards. 13
1.3 The V model of software development. 13
1.4 Airborne future certification framework based on Figure 1.2. 15
1.5 ED-324 W-shape development 15
1.6 Complexity and criticality of aeronautical tasks 17
1.7 Object detection neural networks applied to runway detection (from the LARD dataset Ducoffe et al. [2023]) (a) and obstacle detection Boeing [2025] (b) 18
1.8 Plausible perturbations applied to an airport runway picture. 19
1.9 Overview of the main contributions. 21

2.1 2D example with $x^\star$ and $f(x^\star)$; property verified 26
2.2 Visual example of white-noise perturbation 27
2.3 Visual example of brightness perturbation 28
2.4 2D comparison of brightness and white-noise perturbations 28
2.5 2D example with $\mathcal{C}$ and $f(\mathcal{C}) \subseteq f^a(\mathcal{C})$ 30
2.6 Verified, Violated or Unknown verification property 31
2.7 The Rectified Linear Unit function (ReLU) 32
2.8 Classifier robustness 33
2.9 Linear relaxation of a random non-linear function and a ReLU function 34
2.10 Comparison of the linear relaxation of the ReLU function. 35
2.11 Two standard ReLU relaxations 37
2.12 CROWN DeepPoly ReLU selection 38
2.13 Linear backward bound propagation 39
2.14 Interval Bound Propagation (IBP) 40
2.15 Some interval arithmetic operations. 41
2.16 Constant and linear relaxations of a ReLU function. 41
2.17 CROWN backward pass with pre-activation bound computation 42
2.18 Linear relaxation of a ReLU function for different input intervals 43
2.19 CROWN backward pass with pre-activation bound computation 43
2.20 Illustration of a function node in the PyTorch computational graph. . . 45
2.21 A computational graph 45
2.22 Illustration of a `Bound class` node in `auto-LiRPA` 46
2.23 Inference vs. relaxation pipelines 50
2.24 auto-LiRPA pipeline for 1D brightness perturbations 50
2.25 Wrapper neural network to encode brightness perturbation 51

3.1 runway viewed under increasing rotation angles. 54
3.2 A 5×5 RGB picture. 55

3.3 Inverse rotation applied to find the corresponding location in the original coordinates. . . . 57
3.4 The four nearest neighbour coordinates. . . . 58
3.5 Bilinear interpolation weights 58
3.6 The four nearest neighbors' pixel values. . . . 59
3.7 Illustration of geometric transformations. . . . 60
3.8 The inverse rotation traces a trajectory in the original picture coordinates. 62
3.9 The inverse rotation crosses three interpolation neighborhoods 63
3.10 Heterogeneity of pixel variations 64
3.11 Linear relaxations of pixel values. . . . 65
3.12 Constant relaxations of pixel values 65
3.13 Pipeline for geometric perturbations 65
3.14 Impact of the interval size on relaxation. . . . 66
3.15 Comparison of constant relaxations on single vs. split intervals. . . . 66
3.16 Overview of the verification process. . . . 67
3.17 Verification methods workflow. . . . 68
3.18 Semantify-NN verification workflow. . . . 69
3.19 Coordinate domain 70
3.20 IBP transforms the path into a coordinate domain. . . . 70
3.21 The method calculates distances to all four vertices of the coordinate domain 71
3.22 Distance bounds calculation for FGV. . . . 72
3.23 Contributing pixels 73
3.24 The sampling problem. . . . 75
3.25 Visualizing lipschitz constraints. . . . 76
3.26 Impact of the Lipschitz constant $\mathcal{L}_{i,j}$. . . . 76
3.27 Impact of the sampling distance. . . . 77
3.28 Impact of the sampling density on the admissible region. . . . 77
3.29 Visualizing the three-step strateg 78
3.30 Verification workflow for a single sub-interval. . . . 78
3.31 Detailed workflow of the Lipschitz-based relaxation. . . . 79
3.32 Comparison of Linear Relaxation vs. Piecewise Linear Relaxation. . . . 79

4.1 Comparison of different linear candidates for the upper bound. . . . 84
4.2 Visualizing the tightness of valid candidates by the area of their over-approximation 84
4.3 Residuals corresponding to the candidates in Figure 4.1. . . . 85
4.4 Minimizing the average residual creates a small over-approximation area. 86
4.5 Comparison of the over-approximations (DeepG, PWL, Super-DeepG) 87
4.6 Two linear over-approximation solving ($\mathfrak{P}_{split}$) (DeepG on subsets) . . 88
4.7 Visualization of affine independence for $d = 2$ 90
4.8 Visualisation of a single sub-problem ($\mathfrak{P}^m$) 91
4.9 Visualisation of the valid slope interval. . . . 93
4.10 Illustration of an empty slope interval. . . . 93
4.11 Visualization of the d-dimensional candidate enumeration for $d = 2$. . . 98
4.12 Overview of Step 1. . . . 101
4.13 A 3×3 picture with integer coordinates and pixel values. . . . 104
4.14 The minimal sufficient correction term is the absolute value of the minimum residual. . . . 106
4.15 Determination of $\overline{c}$ using the admissible region. . . . 107

4.16 Visualizing Lipschitz constraints on the residual function 107
4.17 Geometric justification for the error bound. 108
4.18 Determination of $\bar{c}$ with one sample. 109
4.19 First iteration of DeepG algorithm. 110
4.20 Second iteration of DeepG algorithm. 111
4.21 Close-up visualization of the stopping criterion for positive, zero, and negative residuals. 112
4.22 Super-DeepG approach (STEP 3) with N=32 samples. 112
4.23 Overview of Step 3: Guaranteeing mathematical soundness for the linear bounds. 114
4.24 Increasing P beyond 10 brings no significant gains (log scale). Rotation in $[-30°, 30°]$ on MNIST. 121
4.25 Smaller interval sizes improve certified accuracy more than increasing N (rotations in $[-30°, 30°]$ on MNIST shown), log scale. 122

5.1 Single object detection: identifying a runway with a purple bounding box in aerial picture (Enhanced with a zoom for clarity) 123
5.2 Comparison of different Intersection over Union (IoU) values. 124
5.3 Quantitative example of $\text{IoU}_{b_{gt}}(b)$ calculation with predicted bounding box $b = [-2, 1, 0, 2]$ and ground truth $b_{gt} = [0, -1, 2, 1]$. The grid is indexed by integer coordinates centered at $(0, 0)$ in the center of the picture. 125
5.4 Illustration of the relaxation of an object detection neural network. . . 126
5.5 Five examples of bounding boxes included in the set $f^a(\mathcal{C})$. 127
5.6 Correctness for five examples included in $f^a(\mathcal{C})$. The ground truth b_{gt} is fixed (turquoise). 127
5.7 3D visualization of $\text{IoU}_{b_{gt}}(b)$ (z_2, z_3 fixed). 128
5.8 Illustration of the relaxation and verification of an object detection neural network. 128
5.9 Decision tree for selecting a **sound** minimization strategy based on function properties. 130
5.10 Univariate analysis of $\text{IoU}_{b_{gt}}(b)$. Each subplot shows the function's variation when one coordinate is modified and the other three are fixed to their ground truth values. 132
5.11 Variation table of $\text{IoU}_{b_{gt}}(b)$ with respect to z_0. 135
5.12 16 bounding box combinations evaluated at the vertices 136
5.13 Examples of certified accuracy (in %) obtained for some of the tested perturbations for different verification approach combinations and the two use cases. 140
5.14 Impact of a brightness perturbation $|\alpha_b| \leq 0.002$ on a trajectory computed with the CROWN method: orange boxes indicate slight impact, while the red box indicates a strong one (with a guaranteed minimum $\text{IoU}_{b_{gt}}$ value of 0.34). 141
5.15 Computational graph for the $\text{CoIoU}_{b_{gt}}(b)$ metric. 143
5.16 Composition of $\text{CoIoU}_{b_{gt}}$ wrapper layers with the neural network. . . . 143
5.17 Comparison of the inference workflow between coordinate and anchor-based predictions. 145
5.18 Comparison of the verification steps. 146

6.1 Robustness verification workflow 153
6.2 Moving ground truth 157
6.3 Views of a runway under a projective transformation 158

List of Tables

1.1 Picture perturbation families considered in formal verification of neural network robustness. 20

3.1 Matrix representations of the groups and their transformation parameters. 56
3.2 Matrix representations of the geometric transformations considered in this work. 56
3.3 Bilinear interpolation in the integer case $(i', j') = v_k$. 59
3.4 Comparison of state-of-the-art geometric verification tools. 68
3.5 Time to compute interval bounds and network outputs (Semantify-NN and FGV). 74
3.6 Pros and cons of the two bounding strategies 81

4.1 Tools implementing geometric abstract domains 82
4.2 Super-DeepG geometric verifier 83
4.3 Initialization of the priority queue. 110
4.4 Priority queue after one iteration. 110
4.5 Priority queue after two iterations. 111
4.6 Parameter configurations used to guarantee Super-DeepG certified accuracy matches or exceeds DeepG. Configuration numbers in the first column link directly to the results presented in Table 4.7. 117
4.7 Comparison of DeepG and Super-DeepG on PGD networks with equivalent parameters (see Table 4.6). *Results obtained with $P = 10$ samples. 118
4.8 Super-DeepG performs effectively on neural networks trained with CGT 119
4.9 On networks without geometric training. 120
4.10 Comparison with PWL using different configuration hyperparameters. . 121
4.11 On TinyImageNet VNN-Comp'24 neural network, Super-DeepG achieves higher accuracy than FGV in less time. Rotation in $[-5°, 5°]$. 121

5.1 Overview of the input domains around the reference input $x^\star \in \mathbb{R}^{k_0}$ and their respective range of perturbations across the two datasets. v represents a vector of ones. 139
5.2 Comparison of verification performance between our approach (`Unimodal-IoU`) and Raviv et al on white-noise perturbations. *Robust* corresponds to the number of test pictures fulfilling the robustness guarantee property from Equation (5.6) with a threshold $t = 0.5$. 145

List of Publications

- Cohen et al. [2026a] Noémie Cohen, Timothée Fayard, Mélanie Ducoffe, Christophe Gabreau, Claire Pagetti, Xavier Pucel *Super-DeepG- Certified Geometric Robustness*, HSCC/ICCPS 2026
- Cohen et al. [2026b] Noémie Cohen, Christophe Gabreau, Mélanie Ducoffe, Claire Pagetti, Xavier Pucel *Towards the use of formal methods in the certification of vision-based landing systems using high-dimensional ML algorithms*, ERTS 2025
- Cohen et al. [2025] Noémie Cohen, Mélanie Ducoffe, Ryma Boumazouza, Christophe Gabreau, Claire Pagetti, Xavier Pucel and Audrey Galametz *VerifIoU - Robustness of Object Detection to Perturbations*, DASC 2025
- Cappi et al. [2024] Cyril Cappi, Noémie Cohen, Mélanie Ducoffe, Christophe Gabreau, Laurent Gardes, Adrien Gauffriau, Jean-Brice Ginest, Franck Mamalet, Vincent Mussot, Claire Pagetti *How to design a dataset compliant with an ML-based system ODD?*, ERTS 2024
- Elboher et al. [2024a] Yizhak Elboher, Ryma Elsaleh, Omri Isac, Mélanie Ducoffe, Audrey Galametz, Guillaume Povéda, Ryma Boumazouza, Noémie Cohen, Guy Katz, *Robustness Assessment of a Runway Object Classifier for Safe Aircraft Taxiing*, DASC 2024

Chapter 1

Introduction

This document is the result of a PhD project funded by Airbus and the ANRT (Association Nationale pour la Recherche Technologique), as part of a Cifre contract (Industrial Agreement of Training through Research). The objective of this project is to investigate new methods and techniques to verify safety properties of neural network-based systems, and more specifically *object detection neural networks* in the context of civil aviation.

To understand the motivation behind this work we first review the traditional concepts of avionic safety and their requirement-based development. We then explain how data-driven neural networks disrupt these established concepts. Finally, we detail the specific technical challenges addressed in this manuscript to ensure one safety property of *object detection neural networks* applications. Figure 1.1 illustrates the relations between the different concepts, and how this document addresses them.

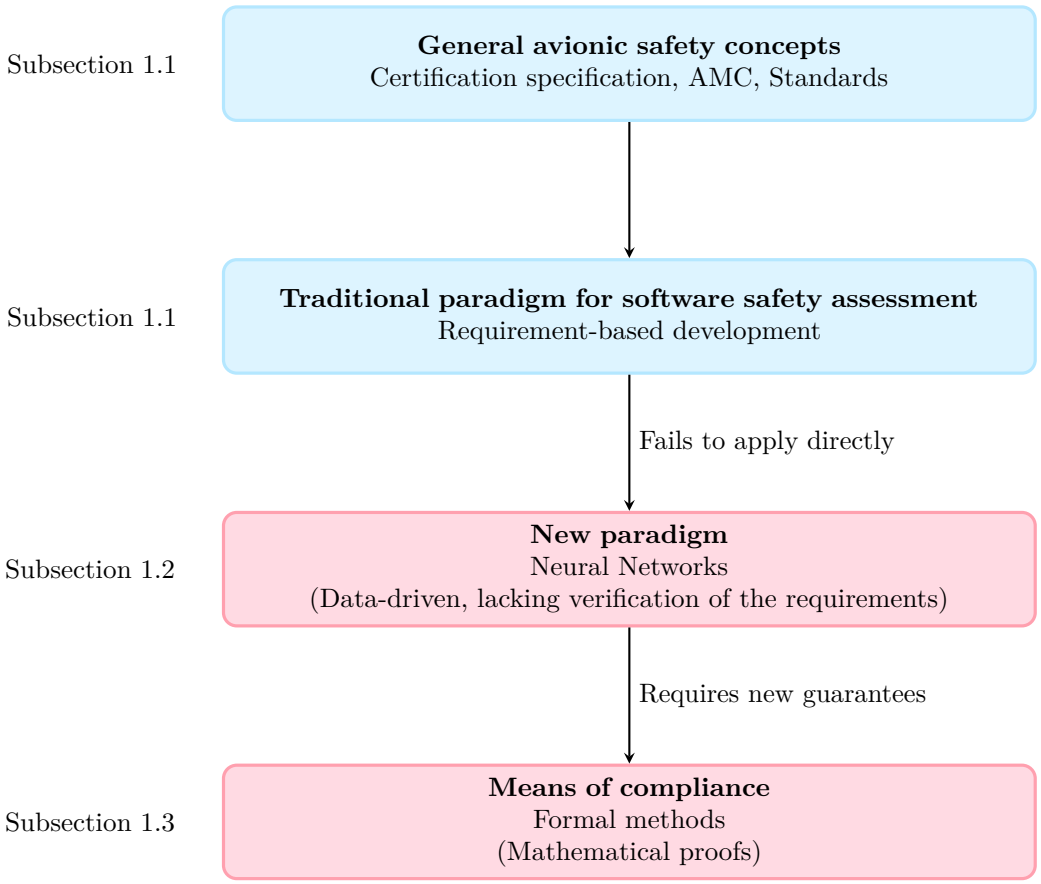


Figure 1.1: Evolution from traditional requirement-based development process to the adoption of formal methods as a means of compliance to ensure the safety of a neural network.

1.1 General safety concepts

Safety-critical systems, i.e. systems with the potential to endanger a person's life, are usually subject to a *certification process*. In practice, any applicant requesting the certification of a system is in charge of convincing a certification authority that their system complies with the regulatory requirements.

Civil aviation safety authorities include the European Aviation Safety Agency (EASA) and the Federal Aviation Administration (FAA). We define below the Certification Specifications (Definition 1.1) and the Acceptable Means of Compliance (Definition 1.2) used to fulfill them. We provide examples (Example 1 and Example 2) in the case of software and hardware development. When the authorities are positively convinced, they deliver a certificate that authorizes the system's operational use.

Definition 1.1 (Certification Specifications (CS))**.** *Certification Specifications are technical rules adopted by the authorities. They do not set exact practices to follow.*

Example 1 (Certification Specifications in Hardware/Software (CS))**.** *The certification specifications for software/hardware systems in avionic safety-related systems are outlined in CS 25.1301 and CS 25.1309 EASA [2018]. In essence, these sections stipulate that an avionic system shall perform its intended function in a safely manner.*

Definition 1.2 (Acceptable Means of Compliance (AMC))**.** *Acceptable Means of Compliance, adopted by the authorities, are approaches for fulfilling the requirements outlined in the CS. In contrast, in the United States, similar guidelines are referred to as Advisory Circulars (AC). (Notations for Europe and the United States documents are separated by a slash (/)). For clarity, the following text focuses on the European documents.*

Example 2 (Acceptable Means of Compliance for Hardware/Software (AMC))**.** *AMC 20-152A RTCA/EUROCAE [2020] and AMC 20-115D RTCA/EUROCAE [2017] are guidelines detailing compliance methods for hardware and software in avionic systems.*

To support applicants in this task, expert committees, composed of industries, certification authorities and academics, have defined standards, guidelines or recommendations (that will be simply referred to as standards, Definition 1.3). These standards are documents that provide high-level certification objectives to be fulfilled.

Definition 1.3 (Standards)**.** *Standards can be viewed as a compilation of good practices and experiences. When standards are recognized as acceptable means of compliance by authorities, then they can be used to demonstrate the conformity to the regulatory requirements. In the United States, standards agencies such as SAE (ARP) and RTCA (DO) are prevalent, whereas EUROCAE (ED) is prominent in Europe.*

Avionic development is structured across two levels of engineering for a given function: *(i)* System/Subsystem, and *(ii)* Item. Figure 1.2 highlights the primary standards for software and hardware in traditional development:

- **At the system level:** ED-79B/ARP4754B SAE/EUROCAE [2010], ED-135/ARP4761A SAE/EUROCAE [2020], ARP5151 SAE [2013] and ARP5150 SAE [2019]
- **At the item level:** ED-80/DO-254 RTCA/EUROCAE [2000] for hardware and ED-12/DO-178C RTCA/EUROCAE [2011] for software.

The standard DO-178C, for instance, provides development assurance guidelines for producing traditional software that performs its intended function with a level of safety confidence compliant with the AMC 20-115D requirements.

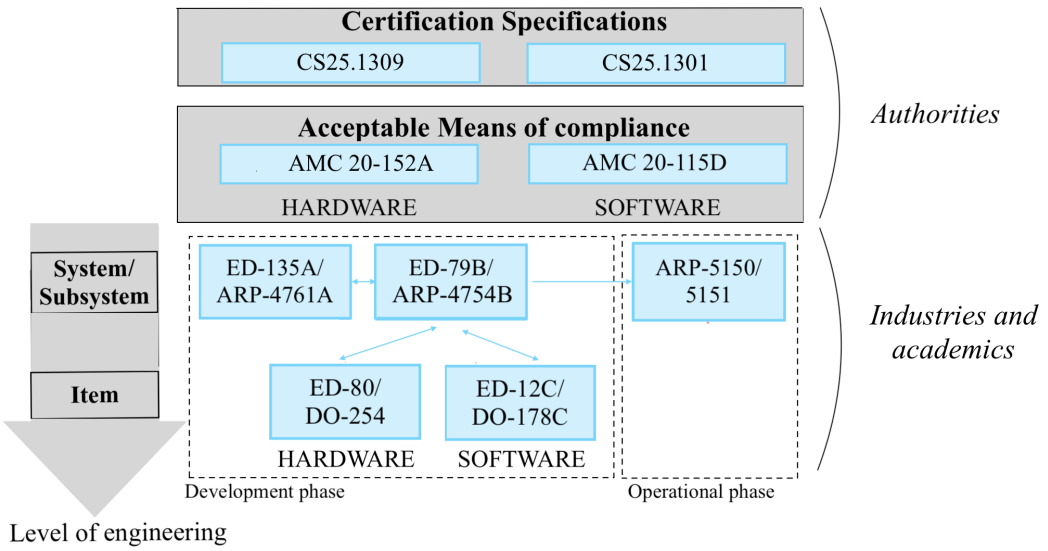


Figure 1.2: Certification specification and related standards.

Today, avionics software development is requirement-based. As described in Figure 1.3, it begins with the allocation of system and safety requirements to software, which then evolve into high-level requirements. An example of this transition is given in Example 3. The software design process (in orange) takes these high-level requirements as input and is in charge of defining the software architecture and the detailed low level requirements that direct source code implementation. Ultimately, the software item undergoes verification through requirement-based testing activities (in purple). The objective is to ensure that the breakdown of requirements is correct and that each development step verifies these requirements. The arrows linking the requirements to the verification activities highlight the traceability needed to prove that every requirement is properly verified.

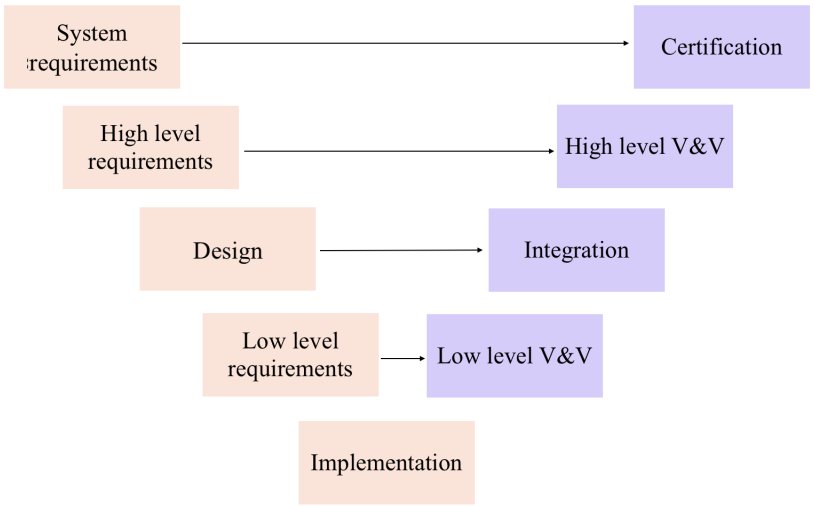


Figure 1.3: The V model of software development.

Example 3 (From system requirement to high-level requirement White and Reza [2012]). *A system-level requirement might state: "The system shall display the total fuel quantity." This is then translated into a more specific software high-level requirement as: "The total fuel quantity shall be displayed in kilograms. The displayed value must*

accurately reflect the real quantity, with a tolerance of ± 0.5%.".

The specific requirements to be verified (the purple activities in Figure 1.3) depend on the **criticality** of the function. Criticality reflects the potential impact of a failure on the aircraft and its occupants (Catastrophic, Hazardous, Major, Minor, No safety effect). This criticality dictates the Design Assurance Level (DAL), a classification that determines the thoroughness of testing and verification processes of the development assurance process required for certification.

Example 4 (Verification based on DAL)**.** *In the DO-178C document, Table A-3 specifies that for DAL A, B and C, the objective "High-level requirements are verifiable." must be satisfied, whereas this is not required for DAL D.*

In summary, traditional engineering relies on requirement-based development to specify, design, and implement software and hardware items. Each item is developed according to its requirements and is verified by requirement-based testing. These activities are based on the assigned DAL. This structured approach requires requirements, design, and code that are understandable to humans and developed consistently. Such a development faces a major obstacle when incorporating machine learning techniques, as the next section explains.

1.2 Impact of ML in the certification method

While traditional engineering relies on requirement-based development, Machine Learning (ML) development employs a data-driven approach. Integrating an ML model into safety-critical systems necessitates a paradigm change in the system development process. This need for change has been highlighted by numerous papers and reports Bhattacharyya et al. [2015]; Varshney [2016]; Jenn et al. [2020]; Boniol et al. [2020]; WG-114/SAE/EUROCAE [2021]; Hawkins et al. [2021]; Burton et al. [2021]; Daedalean and EASA [2021]. In response, the certification authorities and a consortium of experts have proposed new guidelines:

- The EASA provides guidance for the development and deployment of ML-based systems in safety-related applications through the *AI Concept Paper* EASA [2023] (Issues 1 and 2, with Issue 3 forthcoming).
- A joint EU-US standardization group EUROCAE WG-114/SAE G-34 is currently drafting a new standard for the development and certification of aeronautical safety-critical systems based on ML technology (ED-324/ARP6983). *Note that the scope is restricted to ML models trained offline*[1] *with supervised learning and DAL C/D/E.*

Figure 1.4 shows how this new standard integrates into the established avionics concept:

- **No replacement of existing standards.** The new standard does not replace the traditional engineering processes; rather, it acts as a bridge between them. At the highest regulatory level, the EASA Concept Paper is used as the Acceptable Means of Compliance for machine learning development guidance.
- **New engineering level.** The new standard introduces a new level of engineering: the ML-based Constituent (MLC). It operates as an intermediate layer situated directly between the system and the item engineering level.

[1]This means the model is non-adaptive; its parameters do not change after design.

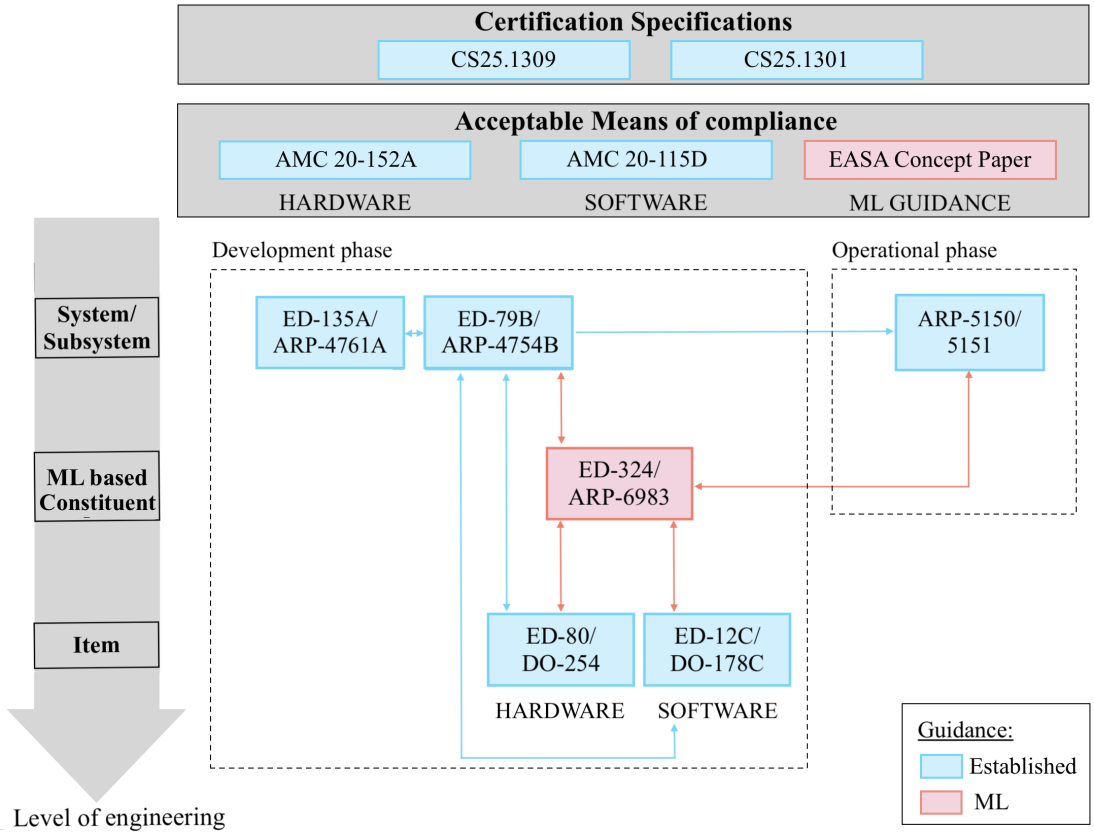


Figure 1.4: Airborne future certification framework based on Figure 1.2.

- **Guide the development process.** The new standard structures the MLC development into two distinct phases (a W-shape development process, Figure 1.5):
  - The first V focuses on data management and the mathematical design of the ML model. The ML model is trained offline. A verification step then ensures the model satisfies its generalization and robustness requirements, which is necessary to ensure that the MLC correctly performs its intended function.
  - The second V focuses on the MLC implementation. It also verifies the MLC behavior on the target platform.

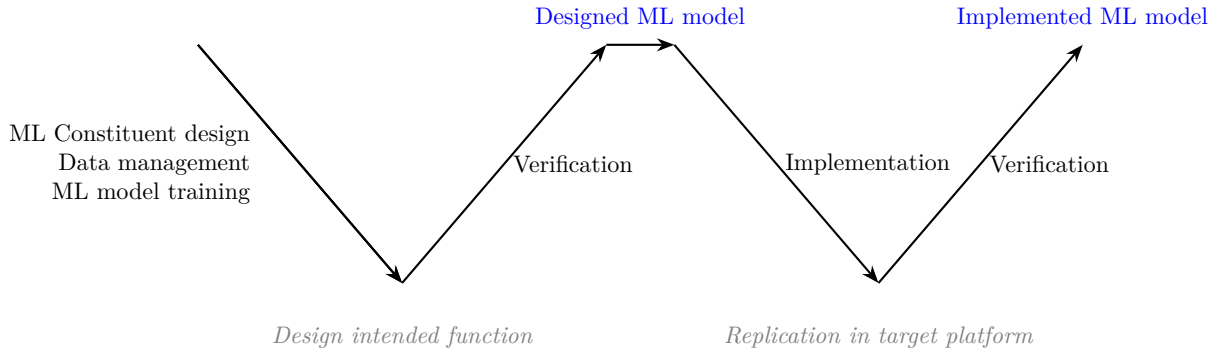


Figure 1.5: ED-324 W-shape development

This W-shape development has a major impact on avionics engineering. The applicant must perform new activities that are not required in traditional engineering. In this thesis,

we focus on the first V (the design of the model) and particularly on the verification that the ML model correctly performs its intended function.

In this thesis, we focus on a specific part of ML: supervised neural network, trained offline. A neural network is a mathematical construct that learns from an offline dataset and applies these learned insights to new, unseen data, extending its capability beyond the specific training examples. Using non-representative data or having insufficient requirements can easily lead to unintended behavior. Among the various challenges this new paradigm introduces to aeronautical certification, one major hurdle is creating an accurate and complete specification for a neural network's intended behavior and subsequently verifying it. This involves answering at least two key questions:

- How do we achieve a correct and comprehensive requirement for the intended behavior of a neural network function?
- How do we verify a neural network against a given requirement?

For the first question, the EASA Concept Paper EASA [2023] and the WG-114 WG-114/SAE/EUROCAE [2021] mention requirements for neural network behavior. The neural network must be:

- **generalizable**. The generalization property guarantees that a neural network achieves the required performance on unseen data.[2]

- **robust**[3]. The robustness property guarantees that a neural network preserves its intended outputs under well-characterized and bounded perturbations to its inputs. In this thesis, we focus on this robustness property.

For the second question, the lack of a means of compliance to verify these generalization and robustness properties poses a significant challenge for aeronautical certification. The following section explains how formal methods offer a viable way to verify the robustness property for critical applications.

1.3 Robustness verification of neural networks

Verifying the robustness requirement poses a major challenge. Traditional testing methods fall short in providing guarantees for neural networks. The following points outline the motivation for employing formal methods:

- **Traditional testing methodology is not adequate for proving the robustness of a neural network.** In traditional aeronautical engineering, testing serves as the primary way to verify the robustness property. This method relies on the concept of equivalence classes EASA [2012]. It partitions the input space into groups where the software is expected to behave the same way for every value. Consequently, testing a few representative points from each group provides sufficient evidence of intended behavior. However, applying this traditional approach to neural networks presents significant challenges. A tiny variation in the input can lead to a sudden and unexpected change in the neural networks output. Researchers often demonstrate this extreme sensitivity through adversarial attacks Szegedy et al. [2013]. This sensitivity makes it very difficult to define reliable equivalence classes for neural networks.

[2] within its intended operational domain, i.e., the specific conditions under which it is designed to function as intended.
[3] While aeronautical standards typically use the term "stability," we use "robustness" to align with the Machine Learning community.

- **Formal methods are a relevant alternative means of compliance compared to massive testing**. For highly critical applications, formal methods are mentioned as a promising means of compliance for verifying robustness. Formal verification is not new to aeronautical engineering; it has already been used in previous Airbus commercial aircraft programs (such as the A380 and A350) Souyris et al. [2009] to prove robustness. A supplement, ED-216/DO-333 RTCA/EUROCAE [2011], is available for employing formal methods as a means of providing evidence that verification objectives are met.
- **Researchers demonstrate the success of formal methods for low-dimensional tasks.** The formal methods for neural networks have proven to be effective in low-dimensional tasks for avionics systems Damour et al. [2021]; Clavière et al. [2021]; Delseny et al. [2021]; Jenn et al. [2020]; Clavière et al. [2022, 2025]; Gabreau et al. [2026]. Figure 1.6 displays various aeronautical applications based on their criticality and complexity, highlighting the EASA roadmap for their certification. As illustrated in this Figure, this progress directly benefits highly safety-critical *physics-based systems* such as equipment health monitoring, advanced sensors, and fuel quantity estimation. While these applications are highly critical, they process lower-dimensional numerical data. Consequently, they are positioned in Zone 2 (Next) of the EASA roadmap. This indicates that the robustness verification of these systems is already becoming achievable, and their certification is expected in the near future.

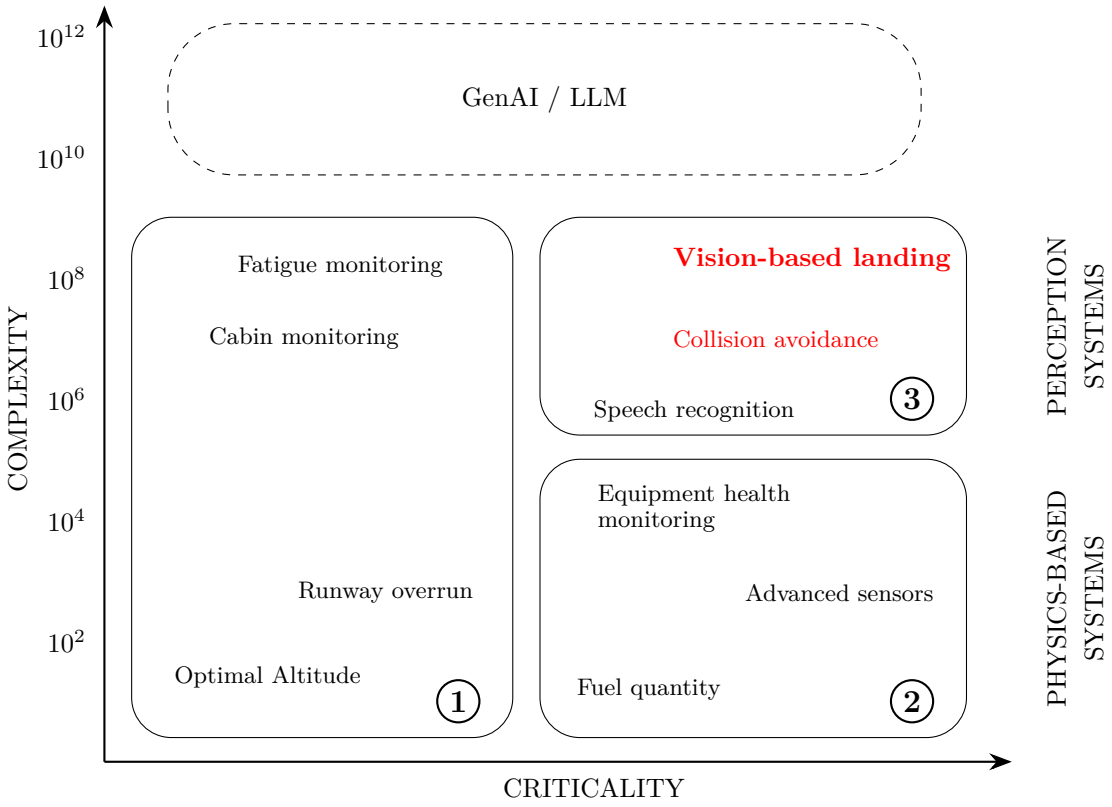


Figure 1.6: Figure taken from the presentation of the article Clavière et al. [2025] based on EASA [2023]. The numbers represent the EASA roadmap timeline: 1 is for current applications (Now), 2 is for upcoming applications (Next), and 3 is for long-term future applications (Future).

The specific challenges addressed in this thesis are the following:

- **The challenge of high-dimensional robustness.** Scaling these methods for

high-dimensional tasks requires a major technological leap. In Figure 1.6, the applications depicted in red are highly safety critical and rely on computer vision, meaning they operate on pictures. These use cases are positioned at the top of the vertical axis, demonstrating their high complexity. Processing pictures poses a high-dimensional challenge, as each pixel acts as a distinct input, and a single picture consists of thousands or even millions of pixels. These critical computer vision applications currently remain in EASA's long-term future roadmap, Zone 3 (Future). Formal verification tools are not yet mature enough to handle high-dimensional pictures for several reasons:

1. **Locality.** Due to the large number of possible pixel combinations, formal verification operates locally, evaluated around a specific reference picture.
2. **Complexity.** Even for well-known perturbations, verifying the neural network requires significant computation time or relies on specialized training procedures.
3. **Perturbation modeling.** Plausible perturbations are difficult to define and model in a format compatible with formal verification tools.
4. **Applicability.** State-of-the-art methods address classification tasks, leaving more advanced computer vision tasks largely unexplored.

This thesis addresses the latter two challenges: Chapters 3 and 4 tackle the perturbation modeling challenge, while Chapter 5 focuses on the applicability challenge. We do not address the first challenge: in every formal verification experiment, we analyze the neighborhood of a single reference picture. The following section introduces the use case we use to guide this work.

1.4 Use case

To ensure that our verification tools address a meaningful problem on the future regulatory roadmap, we chose an application that represents a high-dimensional challenge and is highly critical.

We selected the vision-based landing use case, an application featured in the Autonomous Taxi, Take-Off, and Landing (ATTOL) system developed by Airbus and Onera Acubed [2019]; Airbus and Onera [2020]. As shown in Figure 1.6, it represents a highly complex and highly critical task. The main objective consists of detecting a runway within a picture, as illustrated in Figure 1.7 (a).



(a) (b)

Figure 1.7: Object detection neural networks applied to runway detection (from the LARD dataset Ducoffe et al. [2023]) (a) and obstacle detection Boeing [2025] (b)

This use case has been previously explored in the literature Fischler and Bolles [1981]; Jenn et al. [2020]; Balduzzi et al. [2021]. Furthermore, EASA and Daedalean investigate this vision-based application in their joint reports to prepare the assurance of neural networks in aviation Daedalean and EASA [2021]. Our selection is further supported by access to the LARD dataset Ducoffe et al. [2023], which provides high-dimensional aeronautical pictures of runways.

Technically, runway detection relies on object detection rather than standard classification. This shift introduces a different and more advanced set of challenges for neural networks. Bringing formal verification to this specific task pushes the boundaries of current research while addressing an active industry need. For instance, the Boeing Beacon project depends on comparable object detection techniques to identify obstacles on the runway Boeing [2025] (Figure 1.7 (b)).

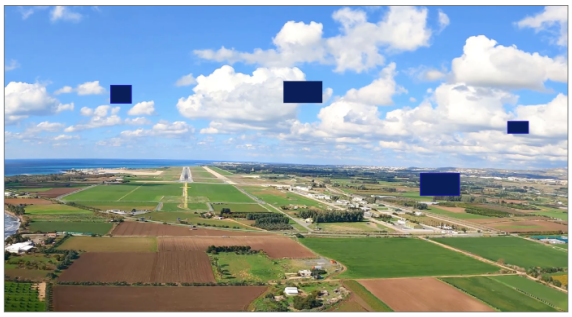
Furthermore, the runway detection system must process pictures that are subject to various environmental and sensor-based perturbations. Depending on the perturbation, each pixel value can vary independently. Many visual changes can happen; we call these changes **perturbations**. Figure 1.8 illustrates four examples of plausible perturbations for an initial input picture. It includes brightness shifts reflecting changes in environmental lighting; scene changes with the introduction of a new object, such as a very fast pigeon passing by; partial occlusions caused by a simulated camera failure (pixel loss) and geometric transformations such as rotations.



(a) Brightness perturbation



(b) Scene perturbation



(c) Partially occluded perturbation



(d) Rotation perturbation

Figure 1.8: Plausible perturbations applied to a picture.

These examples illustrate the wide variety of environmental and operational plausible perturbations that a vision-based landing system must handle. The following section reviews very briefly the state-of-the-art formal methods developed to provide such guarantees against these specific families of perturbations. A more detailed examination of the relevant state-of-the-art will be provided within each contribution section.

1.5 Adopted approach

In the previous section, we introduced the vision-based landing use case and the plausible perturbations that can affect an input picture. To ensure the safety of the neural network, we must verify its robustness against these visual changes. Such verification does not, on its own, guarantee the safety of the entire system, but it represents an important contribution to the broader *certification process*. It is a mandatory step to mitigate *unintended behavior*. This section details the approach we adopt in this thesis to solve this problem. First, we review how existing formal methods handle different families of picture perturbations and explain our focus on geometric transformations. Then, we outline our main results to verify these transformations efficiently and to evaluate the robustness of object detection neural networks.

1.5.1 Choice of perturbations for the robustness property

Much research on the formal robustness of neural networks (on pictures) focuses on white-noise perturbations (l_p-norm) Mangal et al. [2023]. Table 1.1 summarizes the different families of picture perturbations considered in the formal verification of neural networks. When this thesis subject was initially defined, the literature primarily focused on two main types of plausible perturbations beyond standard norm-bounded noise: geometric transformations, brightness and contrast perturbations.

The formal verification field continues to expand. Other studies introduce formal guarantees against specific visual variations, such as convolutional Brückner and Lomuscio [2025] and frequency-domain perturbations Hanspal et al. [2025]. However, addressing these additional perturbations remains outside the scope of this work.

Norm-based, brightness, frequency-domain and contrast perturbations alter pixel values directly while keeping their positions fixed. In contrast, geometric transformations shift and interpolate pixel values, complicating the definition of the perturbation. Existing methods proposed for these transformations were either very computationally efficient but highly imprecise, or very precise but extremely slow. They lacked a practical middle ground required to scale to our specific use case.

Perturbation	**Article**
Norm-bounded pixel perturbations (ℓ_∞, ℓ_2, ℓ_1)	Katz et al. [2017]; Singh et al. [2019]; Wang et al. [2021]; Gehr et al. [2018]; Wang et al. [2018]; Zhang et al. [2018]; Weng et al. [2018]; Ehlers [2017]; Tjeng et al. [2019]; König et al. [2024]; Ducoffe [2022]
Geometric transformations (e.g. rotation)	Mohapatra et al. [2020]; Batten et al. [2024]; Yang et al. [2022]; Balunovic et al. [2019]
Brightness / contrast / bias-field[4] perturbations	Mohapatra et al. [2020]; Elboher et al. [2024b]; Henriksen et al. [2021]
Occlusion perturbations	Mohapatra et al. [2020]; Guo et al. [2023]
Convolutional perturbations (e.g. camera shake)	Brückner and Lomuscio [2025]
Frequency-domain perturbations	Hanspal et al. [2025]

Table 1.1: Picture perturbation families considered in formal verification of neural network robustness.

Additionally, these methods focus on the classification task. Verifying robustness for these computer vision neural networks against complex perturbations raises the following questions:

- First, how do we verify robustness against complex perturbations such as geometric transformations on pictures **efficiently**?
- Second, can we extend the field to other neural network tasks such as object detection?

1.5.2 Methodology and main results

To answer these two questions, we developed new methods to ensure the safety properties required for aeronautical certification. As illustrated in Figure 1.9, we focus on bounding geometric transformations efficiently and evaluating the robustness of an object detection neural network.

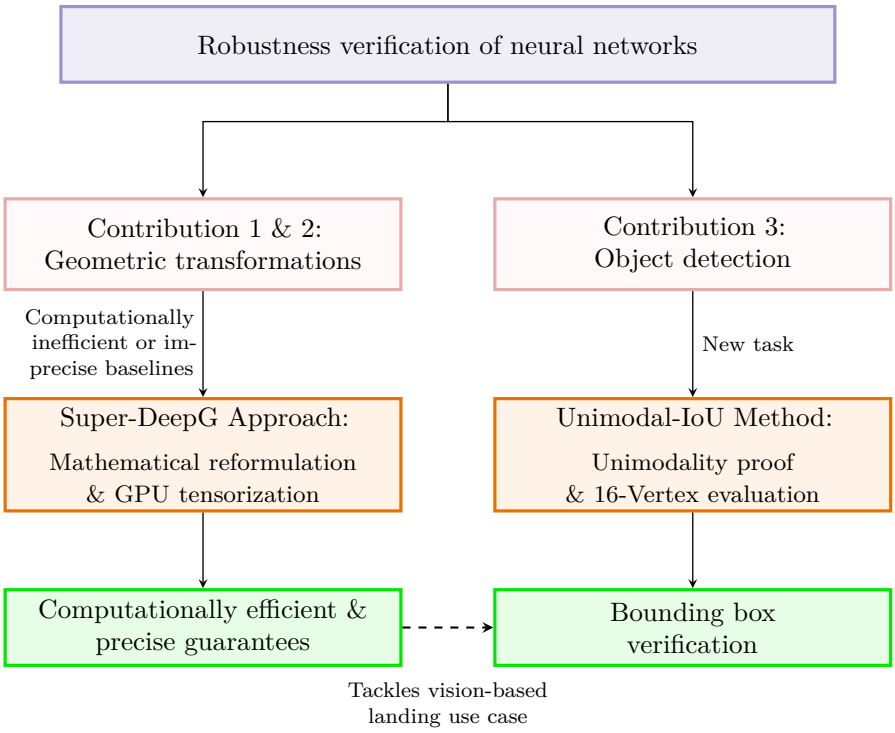


Figure 1.9: Overview of the main contributions.

First, we studied the state of the art for geometric transformations. Existing approaches, such as DeepG, offered high precision but were far too slow to analyze our specific LARD use case. Conversely, other methods were computationally efficient but accumulated too much over-approximation, rendering them too imprecise or required to be used together with a specific training. We aimed to recover the tight precision of DeepG while achieving the speed of GPU computation. To reach this goal, we explored the underlying mathematics and proposed a complete reformulation. We introduce the Super-DeepG approach. In this work, we successfully apply this method to pictures from the LARD dataset for a classification task. By replacing slow linear programming solvers with an analytical solution and a grid-based strategy, we translate the mathematical foundations into parallel tensor operations. This reformulation allows us to evaluate thousands of pixels simultaneously on the GPU. Ultimately, we demonstrate that precision need not be sacrificed for scalability. We show that reformulating computationally

heavy methods for modern parallel hardware makes them highly efficient, offering an interesting path for future research.

Second, to extend formal verification to object detection, we present the Unimodal-IoU method. Evaluating object detection requires verifying bounding box predictions against a ground truth. We analyze the Intersection over Union (IoU) metric and mathematically prove that it is unimodal in each bounding box coordinate. Based on this property, we introduce the Unimodal-IoU method to formally verify bounding box predictions against perturbations without accumulating over-approximation errors.

Overall, we successfully reached our objective to apply both the geometric transformation bounds and the Unimodal-IoU verification to the LARD dataset. While the Super-DeepG approach currently focuses on classification tasks, we have established all the elements to evaluate object detection architectures. Consequently, the method is ready to verify the robustness property on the high-dimensional vision-based landing use case. Due to time constraints, the final step to connect these two domains remains as future work. By bridging the gap between precise mathematical bounds and complex visual features, we meet our objective of providing a scalable verification path for this vision-based landing system.

All our developments are open-source. The codes for both the geometric robustness tool[5] and the object detection evaluation method[6] are publicly available on GitHub.

1.6 Organization of the Manuscript

The manuscript is organized as follows:

- **Part I: Context and Background**
  - The **Introduction** discusses the challenges of integrating Machine Learning (ML) into safety-critical systems that are subject to certification constraints. It presents the main objective of the thesis: adapting formal verification methods for high-dimensional ML systems, such as vision-based landing.
  - The **Background** chapter introduces the concepts of reachability analysis and robustness to evaluate neural networks against input perturbations, such as white-noise, contrast and brightness. It details relaxation techniques, such as Interval Bound Propagation (IBP) and CROWN, which are used to compute an over-approximation of the neural network's outputs.
- **Part II: Geometric Robustness (Contributions 1 and 2)**
  - The **Geometric Robustness** chapter explains how geometric transformations (e.g., rotations, translations) act on pictures and formalizes the associated verification problem. It compares existing verification strategies, highlighting the trade-offs between operator-wise bounding methods and Lipschitz optimization methods.
  - The **Super-DeepG** chapter presents our method and tool. We reformulate the Lipschitz optimization strategy by replacing linear programming solvers with an analytical solution. This new approach allows for GPU parallelization, making the verification of geometric robustness scalable and computationally efficient while maintaining precision.

[5] https://github.com/NoCohen66/superDeepG
[6] https://github.com/NoCohen66/Verification4ObjectDetection

- **Part III: Object Detection Robustness (Contribution 3)**
  - The **Object Detection** chapter extends our formal verification capabilities beyond simple classification to complex object detection tasks. We analyze the Intersection over Union (IoU) metric and mathematically prove that it is unimodal in each bounding box coordinate. Based on this property, we introduce the Unimodal-IoU method to formally verify bounding box predictions against perturbations without accumulating over-approximation errors.
- **Part IV: Conclusion and Perspectives**
  - The **Conclusion** chapter summarizes the results achieved. It also discusses future research perspectives, such as combining geometric transformations with object detection and addressing the massive combinatorial space of real-world plausible perturbations.

Part I

Background

Chapter 2

Background

As highlighted in the introduction (Chapter 1), robustness is a key safety property of neural networks. We motivate and define reachability analysis, formulated in terms of input domain and specification set, which helps us reason about robustness. This chapter begins by establishing the concept of reachability analysis in a general context, applicable to any function. We then specialize it to neural networks and show how linear and constant relaxations are used in practice to prove the robustness of neural networks.

2.1 Verification of reachability properties

This section introduces the concept of a reachability property and explains why, in practice, we must rely on approximation techniques to verify it.

2.1.1 Reachability property

A formal property is a sentence expressed in a formal language. Formal properties can be used to express that a function behaves in a certain way, and in particular as intended Pnueli [1977]. We consider a function $f : \mathbb{R}^p \to \mathbb{R}^m$, an input domain $\mathcal{X} \subseteq \mathbb{R}^p$, and a specification set $Y \subseteq \mathbb{R}^m$, as illustrated in Figure 2.1.

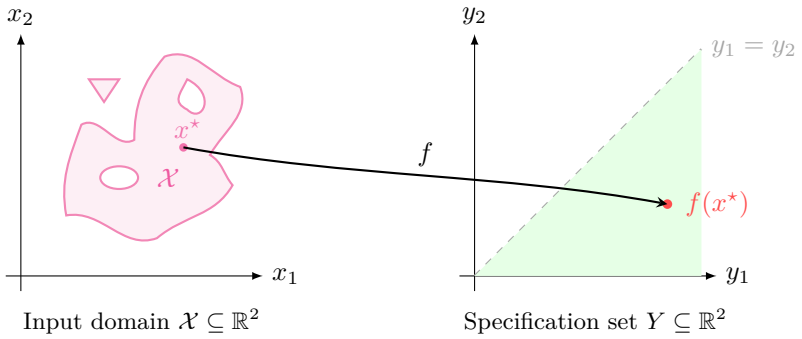


Figure 2.1: 2D example with an input $x^\star \in \mathcal{X}$ and its output $f(x^\star)$. The specification set, in green, is $Y = \{(y_1, y_2) \mid y_1 \geq y_2\}$.

Reachability means that for any input taken from an input domain, here $\mathcal{X}$, the corresponding output is guaranteed to remain within a specification set, here Y.

Definition 2.1 (Reachability property)**.** *Given sets $\mathcal{X}, Y$ and a function f, the (reachable) image of $\mathcal{X}$ under f is $f(\mathcal{X}) = \{f(x) \in \mathbb{R}^m \mid x \in \mathcal{X}\}$. The property of reachability φ_R holds if:*

$$\varphi_R \stackrel{def}{=} f(\mathcal{X}) \subseteq Y$$

Notation 1 (Set image). *We write $f(\mathcal{X})$ to denote the image of a set under f, i.e., $f(\mathcal{X}) = \{f(x) \mid x \in \mathcal{X}\}$, slightly abusing notation by applying a pointwise function directly to a set.*

2.1.2 Input domain approximation

In practice, our analysis focuses on an input domain $\mathcal{X}$ representing local, real-world variations of a known reference input $x^\star$. These variations are called *perturbations*. A widely studied perturbation in the machine learning community is to modify a picture by adding *white noise* (e.g., due to sensor noise or camera faults). Figure 2.2 (b) illustrates an image processing function such as object detection, in which a white noise perturbation can be imperceptible to the human eye. Nevertheless, even such a small alteration to the input can cause the function f to produce an incorrect output, especially in image processing tasks Szegedy et al. [2013]. Robustness analysis aims to verify that the function's output remains correct for small perturbations (case b; Remark 2 gives a typical range for a *small* perturbation). The robustness property in the presence of perturbation is a specific case of the reachability property $f(\mathcal{X}) \subseteq Y$ (Definition 2.1), where the specification set Y is the set of correct outputs, which varies according to the nature of the function f (classification or regression).



(a) Original picture (b) Small white-noise (c) Considerable white-noise

Figure 2.2: An example of a white-noise perturbation. (b) The change is imperceptible to the human eye, making the two pictures appear identical. Picture (c) illustrates a large-magnitude perturbation, where the noise is visually prominent and completely obscures the original content.

In this thesis, we focus on image processing, so the input space is a set of pictures of predefined size. Formally, a picture and the white-noise input domain are mathematically defined in Definition 2.2 and illustrated in Example 5.

Definition 2.2 (Picture). *A picture is represented as a matrix of pixel intensity values, typically in $[p_{min}, p_{max}] = [0, 255]$. Without loss of generality, we normalize pixel values to $[0, 1]$. For a given pixel $p'_{i,j}$ at row i and column j, its normalized value, $p_{i,j}$, is calculated as: $p_{i,j} = \frac{p'_{i,j} - p_{min}}{p_{max} - p_{min}} = \frac{p'_{i,j}}{255}$. A multi-channel picture consists of c matrices of size $n \times m$. This multi-dimensional representation, $x \in \mathbb{R}^{c \times n \times m}$, is also known as a tensor and can be flattened to a vector in $\mathbb{R}^{k_0}$ with $k_0 = c \times n \times m$ when needed.*

Example 5 (White-noise input domain). *The white-noise input domain around a reference input $x^\star \in \mathbb{R}^{k_0}$ consists of all pictures of the same size that are within a distance ϵ of $x^\star$. This distance is measured using the infinity norm ($\|\cdot\|_\infty$):*

$$\mathcal{X}_{noise} = \left\{ x \in \mathbb{R}^{k_0} \;\middle|\; \|x - x^\star\|_\infty \leq \epsilon \right\} = \left\{ x \in \mathbb{R}^{k_0} \;\middle|\; x_i^\star - \epsilon \leq x_i \leq x_i^\star + \epsilon \;\; \forall i = 1, \ldots, k_0 \right\}$$

Each pixel is perturbed independently within the range $[-\epsilon, +\epsilon]$.

Remark 1 (White-noise smallest possible perturbation)**.** *Consider a perturbation* $\epsilon_{min} = 1$ *in the* $[0, 255]$ *discrete scale. On a normalized* $[0, 1]$ *scale, this corresponds to a perturbation of* $\epsilon_{min} = 1/255 \approx 3.9 \times 10^{-3}$*. Since* ϵ_{min} *corresponds to the smallest possible perturbation, any perturbation radius smaller than this value is physically meaningless.*

Remark 2 (Small perturbation)**.** *On a normalized* $[0, 1]$ *scale, a small perturbation is often taken as* $\epsilon \leq 8/255$*, but this depends on the dataset Croce et al. [2020].*

Remark 3 (White-noise input domain size)**.** *Consider a discrete variation for a small* 28×28 *picture with* c *channels, it indeed contains* $k_0 = c \times 784$ *pixels. If each pixel is either unchanged or altered by the smallest possible perturbation value* $+\epsilon_{min}$ *or* $-\epsilon_{min}$*, the number of possible pictures is* 3^{k_0}*. This results in approximately* 10^{374} *and* 10^{1122} *combinations for 1 and 3 channels, respectively. Consequently, a tiny perturbation radius makes exhaustive tests computationally intractable.*

Another real-world perturbation is the brightness perturbation, illustrated in Figure 2.3. Brightness perturbations differ from white-noise perturbations because all pixels are shifted by the *same* value.



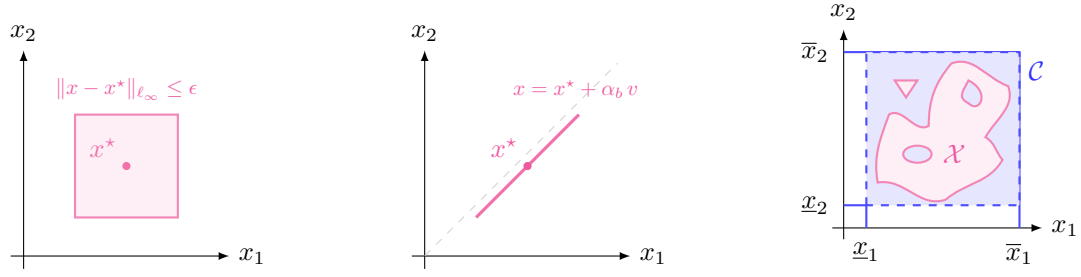
(a) Original picture (b) Increased Brightness (c) Decreased Brightness

Figure 2.3: An example of a brightness perturbation. The original picture (a) is shown with uniformly increased (b) and decreased (c) brightness.

Example 6 (Brightness input domain)**.** *The brightness input domain around an input* $x^\star \in \mathbb{R}^{k_0}$ *consists of all inputs of the same size as* $x^\star$ *to which a uniform addition is applied across all pixels using a coefficient* $\alpha_b \in \mathbb{R}$ *in a range* $[-\epsilon, +\epsilon]$*:*

$$\mathcal{X}_{bright} = \left\{ x \in \mathbb{R}^{k_0} \;\middle|\; x = x^\star + \alpha_b\, v,\;\; \alpha_b \in [-\epsilon, +\epsilon],\;\; v = (1, \ldots, 1) \in \mathbb{R}^{k_0} \right\}.$$

Figure 2.4 (a) (b) illustrates these white-noise and brightness input domains in 2D.



(a) White-noise domain $\mathcal{X}_{\text{noise}}$ (b) Brightness domain $\mathcal{X}_{\text{bright}}$ (c) Complex domain $\mathcal{X}$ and $\mathcal{C}$

Figure 2.4: Visualizations of different input domains in a 2D input space, representing two pixels. (a) White-noise domain, allowing for an independent perturbation of up to $\epsilon = 0.8$ on each input component. (b) Brightness domain, applying a perturbation $\alpha_b \in [-0.8, 0.8]$ to all components. (c) A complex domain $\mathcal{X}$ over-approximated by an interval box $\mathcal{C}$.

The input domain $\mathcal{X}$ can represent more complex perturbations than the brightness or white-noise domains, as illustrated in Figure 2.4 (c). The literature Xiang et al. [2017] usually assumes that it is over-approximated by a convex shape $\mathcal{C}$ where $\mathcal{X} \subseteq \mathcal{C}$. The simplest choice is to model the input domain as an *interval box*. This shape is simple because it only defines an independent range for each dimension:

$$\mathcal{C} = \left\{ x \in \mathbb{R}^p \;\middle|\; \underline{x_i} \leq x_i \leq \overline{x}_i \;\; \forall i \in \{1, \ldots, p\} \right\} \tag{2.1}$$

Here, for each dimension i, the values $\underline{x_i}$ and $\overline{x}_i$ are pre-defined constants representing the lower and upper bounds of the input domain.

More generally, any input domain $\mathcal{X}$ can be over-approximated by a convex shape $\mathcal{C} \supseteq \mathcal{X}$, such as an interval box. This approximation strategy relies on the implication that if the property holds for the larger set $\mathcal{C}$, it necessarily holds for the original input domain $\mathcal{X}$ (see Property 2.1).

Remark 4 (White-noise and brightness as interval box)**.** *The white-noise perturbation is already modeled as an interval box $\mathcal{X}_{noise} = \mathcal{C}_{noise}$, where $\underline{x_i} = x_i^\star - \epsilon$ and $\overline{x_i} = x_i^\star + \epsilon$. The brightness perturbation domain can be bounded by (or over-approximated by) the white-noise input domain, such that $\mathcal{X}_{bright} \subseteq \mathcal{C}_{noise}$. For instance, the domain in Figure 2.4 (b) is encapsulated by the domain in Figure 2.4 (a).*

Property 2.1 (Reachability property on an over-approximated $\mathcal{X}$)**.** *Given sets $\mathcal{C} \supseteq \mathcal{X}, Y$ and a function f. The reachability property $\varphi_{\mathcal{C}R}$ is satisfied if the (reachable) image of $\mathcal{C}$ under f, $f(\mathcal{C})$ is included in the set Y: $\varphi_{\mathcal{C}R} \overset{def}{=} f(\mathcal{C}) \subseteq Y$. Hence,*

$$\varphi_{\mathcal{C}R} \implies \varphi_R$$

When the input domain $\mathcal{X}$ is complex (see Figure 2.4) computing its image can be difficult or impossible, which prevents a direct verification of the reachability property. In this case, one solution from the literature is to define an over-approximation $\mathcal{C}$ of the input domain and compute $f(\mathcal{C}) = \{f(x) \in \mathbb{R}^m \mid x \in \mathcal{C}\}$ to invoke Property 2.1.

2.1.3 Computation of the (reachable) image of $\mathcal{C}$ under f

Let $\mathcal{C}$ be a convex shape over-approximating $\mathcal{X}$ (i.e., $\mathcal{C} \supseteq \mathcal{X}$). We now focus on the difficulty of characterizing $f(\mathcal{C})$. We consider two cases:

- **f is a linear (affine) function.** Let $f : \mathbb{R}^p \to \mathbb{R}^m$ be an affine function of the form $f(x) = Ax + B$ with coefficients $A \in \mathbb{R}^{m \times p}$, $B \in \mathbb{R}^m$, and input $x \in \mathbb{R}^p$. If $\mathcal{C}$ is an interval box, then the (reachable) image of $\mathcal{C}$ under f $f(\mathcal{C}) = \{\, Ax + B \mid x \in \mathcal{C} \,\}$ is a zonotope, the image of an interval box under an affine function Girard [2005].

  Definition 2.3 (Zonotope)**.** *A zonotope Z is a set such that*

  $$Z = \left\{ y \in \mathbb{R}^m \;\middle|\; y = c + \sum_{i=1}^{k} \xi_i g_i, \;\; -1 \leq \xi_i \leq 1 \right\}, \text{ where } c, g_1, \ldots, g_k \in \mathbb{R}^m$$

  In practice, the (reachable) image of $\mathcal{C}$ under a linear (affine) function f, $f(\mathcal{C})$ is efficiently computable via basic linear algebra.

- f **is a non-linear function.** By contrast, for non-linear f, computing $f(\mathcal{C})$ exactly is intractable or too costly in general (it is an NP-hard problem Katz et al. [2017]), even when $\mathcal{C}$ is an interval box. We therefore rely on *over-approximations* of $f(\mathcal{C})$, denoted by $f^a(\mathcal{C})$ where f^a is a *relaxation* of f. While this term often refers to the process of constructing bounds, we adopt it, for simplicity, to also describe the resulting over-approximating function f^a, formally known as an abstract transformer Singh et al. [2019].

  Definition 2.4 (Relaxation Singh et al. [2019])**.** *Let $f : \mathbb{R}^p \to \mathbb{R}^m$ be a function and $\mathcal{C}$ be an input domain. A relaxation of f for the input domain $\mathcal{C}$ is a function $f^a : \mathbb{R}^p \to \mathbb{R}^m$ which satisfies:*

$$f(\mathcal{C}) \subseteq f^a(\mathcal{C})$$

If f is a composition of functions (whether linear or non-linear), a relaxation of f can be constructed by composing relaxations of each function, as detailed in Property 2.2.

Property 2.2 (Composition of approximations Yang et al. [2021])**.** *Given an input domain $\mathcal{C}$, and a function $f(x) = f_L \circ f_{L-1} \circ \cdots \circ f_1(x)$. A (reachable) image of $\mathcal{C}$ under a relaxation of f, $f^a(\mathcal{C})$ can be computed by composing linear or constant relaxations (detailed in the following sections) of all functions f_i^a $\forall i \in \{1, \ldots, L\}$:*

$$f^a(\mathcal{C}) = \{f_L^a \circ f_{L-1}^a \circ \cdots \circ f_1^a(x) \mid x \in \mathcal{C}\} \tag{2.2}$$

Having over-approximated $f(\mathcal{C})$ by $f^a(\mathcal{C})$ as in Figure 2.5, we can now define an over-approximated reachability property.

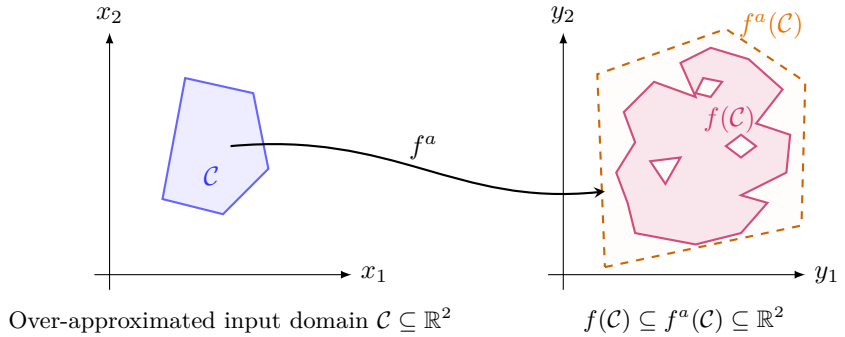


Figure 2.5: Left: an input domain $\mathcal{C}$. Right: the (reachable) image of $\mathcal{C}$ under f, $f(\mathcal{C})$ (purple) and an enclosing over-approximation $f^a(\mathcal{C})$ (orange, dashed), with $f(\mathcal{C}) \subseteq f^a(\mathcal{C})$.

Property 2.3 (Over-approximated reachability property)**.** *The property of reachability φ_{RA} is satisfied if the (reachable) image of $\mathcal{C}$ under a relaxation of f, $f^a(\mathcal{C})$ is included in the specification set Y: $\varphi_{aCR} \stackrel{def}{=} f^a(\mathcal{C}) \subseteq Y$. Hence,*

$$\varphi_{aCR} \implies \varphi_{CR} \implies \varphi_R$$

2.1.4 Trade-offs in over-approximations

While computing this over-approximation $f^a(\mathcal{C})$ may be more tractable than determining $f(\mathcal{C})$ or computing the over-approximation $\mathcal{C}$ is more tractable than determining $\mathcal{X}$, these approaches introduces a trade-off between precision and complexity. Specifically, when checking if the property holds, we can encounter three distinct outcomes:

- Verified, if $f^a(\mathcal{C}) \subseteq Y$.

- Violated, if $f^a(\mathcal{C}) \cap Y = \varnothing$; then all reachable outputs lie outside Y.
- Unknown, otherwise. This result occurs when $f^a(\mathcal{C})$ overlaps both Y and its complement. The approximation is too loose to give a definite answer.

Figure 2.6 shows three over-approximations $f^a(\mathcal{C})$ for different input domains.

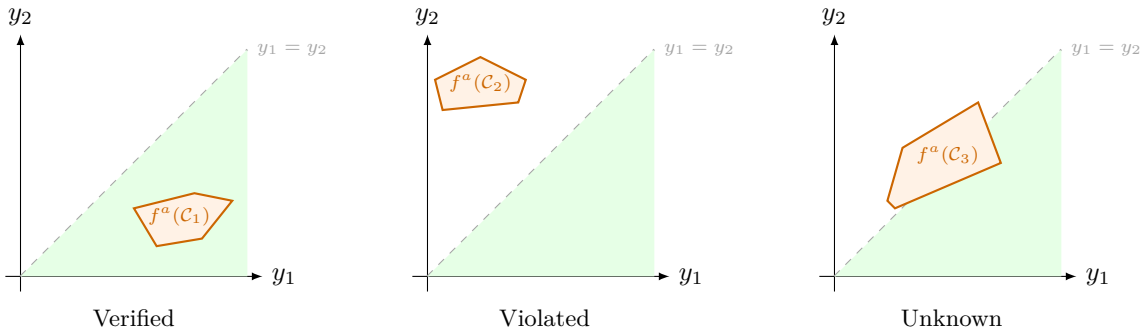


Figure 2.6: Illustration of the three possible outcomes in over-approximated reachability analysis. The specification set is $Y = \{y \mid y_1 \geq y_2\}$.

In our setting, we consider *Unknown* as a failure to verify robustness, since the approximation does not allow us to guarantee robustness (we cannot deduce if the real image $f(\mathcal{X})$ lays entirely in Y or not). In the context of incomplete verification methods (i.e., methods that rely on over-approximation), the more precise the abstraction, the fewer *Unknown* cases during verification.

We have now established a general theoretical framework for verifying reachability through over-approximation in a general case where f can be any function. To apply these verification principles to the specific case of feed-forward neural networks, we first formally define a neural network f as a sequence of linear and non-linear layers, and demonstrate how to verify the robustness of an picture classification function.

2.2 Sequential feed-forward neural networks

A direct sequential feed-forward neural network can be viewed as a function mapping an input (e.g. a picture) to an output (e.g. a classification or an object detection result).

2.2.1 Definition

A sequential feed-forward neural network is built as a composition of layers. Mixing *linear* and *non-linear* layers is essential, since any composition of purely linear layers remains linear and thus cannot approximate complex functions Cybenko [1989]. Throughout this work, we restrict our attention to the *inference* phase (as opposed to the *learning* phase) of networks with **sequential** (acyclic) architectures. We analyze these neural networks during their inference phase, meaning their architecture and parameters (weights and biases) are considered fixed. For simplicity of notation, the input picture $x \in \mathbb{R}^{c \times n \times m}$ is flattened to a vector in $\mathbb{R}^{k_0}$, where $k_0 = c \times n \times m$. Formally, if we denote by f_i the operation performed by the i-th layer, then a neural network with L layers implements a function:

$$f : \begin{cases} \mathbb{R}^{k_0} \to \mathbb{R}^{k_L}, \\ x \mapsto f_L \circ f_{L-1} \circ \cdots \circ f_1(x). \end{cases} \tag{2.3}$$

Definition 2.5 (Linear layer)**.** *A linear (affine) layer, with weight* $A_i \in \mathbb{R}^{k_i \times k_{i-1}}$ *and bias* $B_i \in \mathbb{R}^{k_i}$ *is defined as:*

$$f_i : \begin{cases} \mathbb{R}^{k_{i-1}} \to \mathbb{R}^{k_i}, \\ x \mapsto A_i \cdot x + B_i. \end{cases} \tag{2.4}$$

Remark 5 (Convolutional layer)**.** *Convolution layers are widely used in computer vision to extract features while reducing the large spatial dimensionality of pictures. As shown in Ma and Lu [2017], any convolution layer can be expressed as a linear layer.*

Remark 6 (Average pooling layer)**.** *Average pooling layers partition an input matrix into a grid of windows. For each window containing* k *values* $\{x_1, \ldots, x_k\}$*, the output is the mean* $y = \frac{1}{k} \sum_{i=1}^{k} x_i$*. This operation is linear, since each output is a linear combination of the inputs.*

Remark 7 (Batch normalization layer)**.** *At inference time, a batch normalization layer acts as a linear normalization layer with fixed parameters.*

Unless specified otherwise, we treat all layers other than affine ones (dense, convolution, average pooling, and batch normalization at inference) as non-linear in general (e.g., max-pooling, softmax, and standard activation functions). Among all non-linear activation functions, the *rectified linear unit (ReLU)* is the most widespread in machine learning.

Definition 2.6 (ReLU layer)**.** *The rectified linear unit (ReLU) function is defined by* $\sigma(t) = \max(0, t) \in \mathbb{R}, \quad \forall t \in \mathbb{R}$.

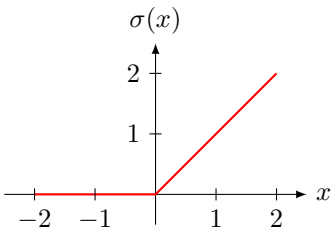


Figure 2.7: The Rectified Linear Unit function (ReLU)

A ReLU layer applies this ***non-linear*** *function element-wise to its input,* $\forall x \in \mathbb{R}^{k_{i-1}}$*:*

$$f_i(x) = (\sigma(z_1), \ldots, \sigma(z_{k_{i-1}})) \in \mathbb{R}^{k_{i-1}}.$$

2.2.2 Example of a reachability property on classification

Classification is a common task in computer vision using neural networks. A simple version is binary classification, where the goal is to choose between two options, for example, answering the question: "Is there a cat or not in the picture?". In its general setting, multi-class classification handles several classes at once. A famous example is the MNIST dataset, which contains pictures of handwritten digits. In this case, there are 10 possible classes, corresponding to the digits 0 through 9.

Let $f : \mathbb{R}^p \to \mathbb{R}^{k_L}$ be a classifier, where k_L is the number of classes. For an input $x \in \mathbb{R}^p$, we denote the vector of output scores as $f(x) = (y_1(x), \ldots, y_{k_L}(x))$. The predicted class corresponds to the index of the highest score. Let $x^\star$ be a reference input with ground-truth class index $gt \in \{1, \ldots, k_L\}$. Let $\mathcal{C} \subseteq \mathbb{R}^p$ be an input domain around $x^\star$ and assume that gt remains the ground-truth class for all $x \in \mathcal{C}$. The classifier is

robust on $\mathcal{C}$ if, for every $x \in \mathcal{C}$, the score of the ground-truth class is greater than all other scores:

$$y_{gt}(x) > y_j(x), \quad \forall x \in \mathcal{C}, \quad \forall j \in \{1, \ldots, k_L\} \setminus \{gt\}.$$

From the reachability viewpoint, the specification set for class gt is the set of all score vectors where the gt-th component is the unique maximum:

$$Y_{gt} = \left\{ y \in \mathbb{R}^{k_L} \;\middle|\; y_{gt} > y_j, \quad \forall j \in \{1, \ldots, k_L\} \setminus \{gt\} \right\}.$$

Let f^a be a relaxation of f. If we can prove that $f^a(\mathcal{C}) \subseteq Y_{gt}$, then by Property 2.1, the reachability property is verified. In Figure 2.8 the binary classifier determines whether a picture contains a runway: *runway* (y_1) or *no runway* (y_2). The reference input $x^\star$ is a picture known to contain a runway, making *runway* the ground-truth class ($gt = 1$). Y_{gt} is defined by the inequality $y_1 > y_2$. Under perturbations within $\mathcal{C}$, the highest-scoring class must remain y_1.

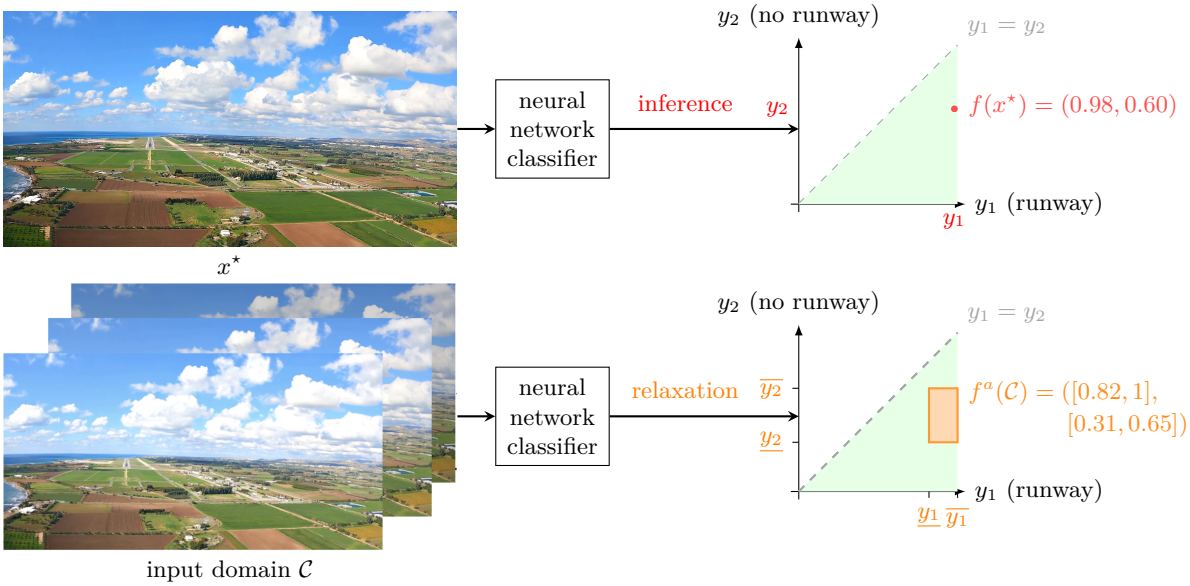


Figure 2.8: Illustration of classifier robustness. Top: inference for a single picture $x^\star$. Bottom: Successful robustness verification over $\mathcal{C}$. The specification set, in green, is $Y = \{(y_1, y_2) \mid y_1 \geq y_2\}$.

In the bottom plot of Figure 2.8, $f^a(\mathcal{C}) \subseteq Y_{gt}$, the orange box is contained entirely within the green specification set Y_{gt}. While this visual inspection is illustrative, in practice, we check the interval bounds of $f^a(\mathcal{C})$ as explained in Remark 8.

Remark 8 (Verifying robustness with $f^a(\mathcal{C})$)**.** *Consider the resulting reachable image $f^a(\mathcal{C})$ defined as follow: $f^a(\mathcal{C}) = \left\{ y \in \mathbb{R}^{k_L} \;\middle|\; \underline{y}_j \leq y_j \leq \overline{y}_j, \; \forall j = 1, \ldots, k_L \right\}$. It represents all possible outputs score as intervals for any input in $\mathcal{C}$. Sufficient condition to check the property: if the lower bound of the ground-truth score is strictly greater than the maximum upper bound of any other score: $\underline{y}_{gt} > \max_{j \neq gt} \overline{y}_j$, then robustness is ensured. In the bottom plot of Figure 2.8, interval bounds are $y_1 \in [0.82, 1]$ and $y_2 \in [0.31, 0.65]$. The check $\underline{y}_{gt} = \underline{y}_1 > \overline{y}_2$ becomes $0.82 > 0.65$, which is true, thus verifying the robustness property.*

We introduce the following notation for componentwise inequalities between vectors. This notation is used throughout the manuscript.

Notation 2. *For $a = (a_1, \ldots, a_d)$ and $b = (b_1, \ldots, b_d)$ in $\mathbb{R}^d$, we write $a \leq b$ if $a_i \leq b_i$ for every $i = 1, \ldots, d$. In other words, vector inequalities are understood componentwise.*

2.3 LiRPA: Neural networks' linear relaxation

Since a neural network f is a composition of linear and non-linear layers, verifying the reachability property introduced in Section 2.1 requires methods to relax layers and compose their approximations. In this subsection, we focus on a standard approximation approach: linear relaxation.

2.3.1 Definition of linear relaxation

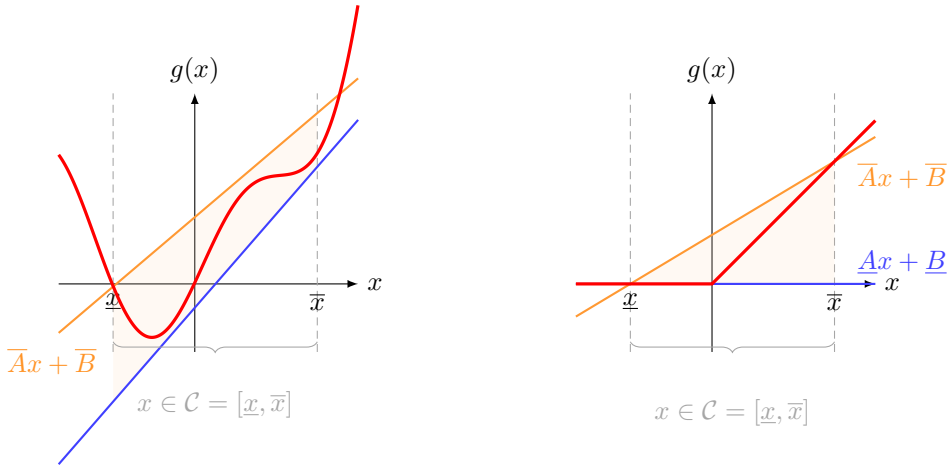
We first introduce the general concept of linear relaxation. Our objective is to construct $f^a(\mathcal{C})$, it is a set of linear bounds $(\underline{f^a}, \overline{f^a})$ for the entire neural network f over the input domain $\mathcal{C}$. This is achieved by computing a relaxation for each individual layer $(\underline{f_i^a}, \overline{f_i^a})$ and composing these relaxations.

Definition 2.7 (Linear relaxation)**.** *Given a domain $\mathcal{C} \subseteq \mathbb{R}^p$ and a linear relaxation of a function $g : \mathbb{R}^p \to \mathbb{R}^m$ is a tuple $(\underline{A}, \overline{A}, \underline{B}, \overline{B})$ such that:*

$$\forall x \in \mathcal{C}, \qquad \underline{A}x + \underline{B} \;\leq\; g(x) \;\leq\; \overline{A}x + \overline{B}$$

The weights $\underline{A}, \overline{A} \in \mathbb{R}^{m \times p}$ and biases $\underline{B}, \overline{B} \in \mathbb{R}^m$ are the coefficient of the linear relaxation. Note that inequalities are applied element-wise.

Figure 2.9 illustrates the linear relaxation for a 1D case: both input and output are scalars (i.e., $m = p = 1$).



$\underline{A}x + \underline{B}$

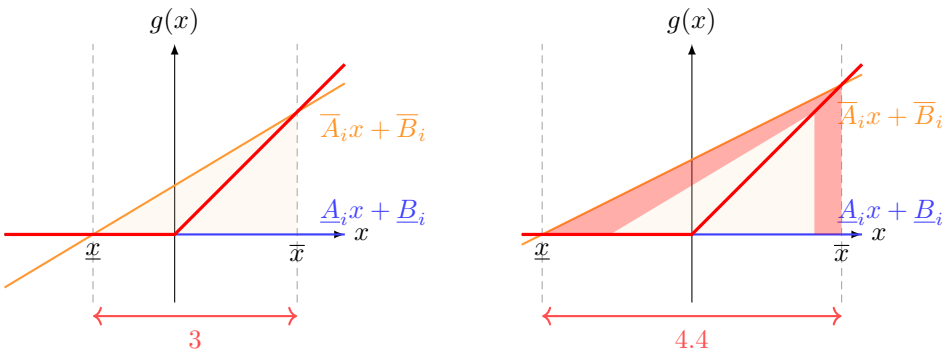
(a) Random function
$1.15x + -0.35 \;\leq\; g(x) \;\leq\; 0.85x + 0.98$

(b) ReLU function
$0x + 0 \;\leq\; g(x) \;\leq\; 0.6x + 0.72$

Figure 2.9: Linear relaxation on $\mathcal{C} = [\underline{x}, \overline{x}]$. For both functions, $g(x)$ (red) is enclosed between an upper affine bound $\overline{A}x + \overline{B}$ (orange) and a lower affine bound $\underline{A}x + \underline{B}$ (blue). The orange shaded band shows the approximation region of the relaxation.

Remark 9 (Impact of the input domain size on precision)**.** *The quality of the linear relaxation depends heavily on the size of the input domain $\mathcal{C} = [\underline{x}, \overline{x}]$. As illustrated in Figure 2.10, when the input domain is larger, the gap between $\underline{A}x + \underline{B}$ and $\overline{A}x + \overline{B}$ increases, resulting in a larger over-approximation area.*

Based on the general definition of linear relaxation (Definition 2.7), we introduce the notation $(\underline{f^a}, \overline{f^a})$ for the relaxation of the network f, and $(\underline{f_i^a}, \overline{f_i^a})$ for the relaxation of a layer f_i.



(a) $\mathcal{C} = [\underline{x}, \overline{x}] = [-1.2, 1.8]$ (b) $\mathcal{C} = [\underline{x}, \overline{x}] = [-2.2, 2.2]$

Figure 2.10: Comparison of the linear relaxation of the ReLU function for a small (a) vs. a larger (b) input domain $\mathcal{C}$. The additional over-approximation due to the wider interval is colored in red.

Notation 3 (Relaxation for the neural network)**.** *A relaxation for the entire network f over the initial input domain $\mathcal{C} \subseteq \mathbb{R}^{k_0}$ is a tuple $(\underline{f^a}, \overline{f^a})$ such that for each x in $\mathcal{C}$:*

$$\underline{f^a}(x) \leq f(x) \leq \overline{f^a}(x) \quad \textit{where} \quad \begin{array}{ll} \underline{f^a}(x) = \underline{A}x + \underline{B} & \underline{A} \in \mathbb{R}^{k_L \times k_0}, \underline{B} \in \mathbb{R}^{k_L} \\ \overline{f^a}(x) = \overline{A}x + \overline{B} & \overline{A} \in \mathbb{R}^{k_L \times k_0}, \overline{B} \in \mathbb{R}^{k_L} \end{array}$$

We denote $f^a(\mathcal{C})$ the entire relaxation for the neural network.

Notation 4 (Relaxation for a layer)**.** *A relaxation for a layer f_i over its input domain $\mathcal{C}_{i-1} \subseteq \mathbb{R}^{k_{i-1}}$ is a tuple $(\underline{f_i^a}, \overline{f_i^a})$ such that for each x in $\in \mathcal{C}_{i-1}$:*

$$\underline{f_i^a}(x) \leq f_i(x) \leq \overline{f_i^a}(x) \quad \textit{where} \quad \begin{array}{ll} \underline{f_i^a}(x) = \underline{A}_i x + \underline{B}_i & \underline{A}_i \in \mathbb{R}^{k_i \times k_{i-1}}, \underline{B}_i \in \mathbb{R}^{k_i} \\ \overline{f_i^a}(x) = \overline{A}_i x + \overline{B}_i & \overline{A}_i \in \mathbb{R}^{k_i \times k_{i-1}}, \overline{B}_i \in \mathbb{R}^{k_i} \end{array} \tag{2.5}$$

Neural networks consist of a sequence of layers, where each layer implements a simple function. Therefore, computing a linear relaxation for the entire network f relies on relaxing each layer f_i individually and composing the resulting approximations. Consequently, constructing a tight global relaxation involves three key steps, performed for every layer f_i:

- Determination of a tight input domain $\mathcal{C}_{i-1}$. The goal is to minimize the over-approximation error described in Remark 9.
- Computation of the linear coefficients $(\underline{A}_i, \overline{A}_i, \underline{B}_i, \overline{B}_i)$ over $\mathcal{C}_{i-1}$ to bound the function f_i.
- Composition of the layer-wise relaxations to build the global approximation.

The following subsections detail these steps, followed by a presentation of standard algorithms (IBP, CROWN, and CROWN-IBP) that implement this procedure efficiently.

2.3.2 Computing the input domain of a layer

In this subsection, we focus on defining $\mathcal{C}_{i-1}$, the input domain of layer f_i (Notation 4).

An important property (detailed in Subsection 2.3.4) is that the composition of linear relaxations is also a linear relaxation. Let the coefficients $\underline{A}', \overline{A}' \in \mathbb{R}^{k_{i-1} \times k_0}$ and $\underline{B}', \overline{B}' \in \mathbb{R}^{k_{i-1}}$ define the linear relaxation of the function $f_{i-1}^a \circ \cdots \circ f_1^a$, representing the composition of the over-approximations of the first $i-1$ layers. We define $\mathcal{C}_{i-1}$ as

the relation between the input domain $\mathcal{C}$ and the over-approximation output of this sub-network:

$$\mathcal{C}_{i-1} = \left\{(x, y) \in \mathcal{C} \times \mathbb{R}^{k_{i-1}} \;\middle|\; \underline{A}'x + \underline{B}' \leq y \leq \overline{A}'x + \overline{B}'\right\} \tag{2.6}$$

However, the relational Equation (2.6) is not directly tractable for all types of layers.

Property 2.4 (Input domain $\mathcal{C}_{i-1}$ of a linear layer)**.** *If $f_i(x) = A_i x + B_i$, the linear relaxation is exact and holds for the set $\mathcal{C}_{i-1}$ defined in Equation* (2.6)*.*

However, for non-linear layers (such as activation functions), finding the coefficients $(\underline{A}_i, \overline{A}_i, \underline{B}_i, \overline{B}_i)$ from Equation (2.5) is an optimization problem. To simplify it, most standard methods consider that the input of any non-linear layers is an interval box called the **pre-activations bounds** Botoeva et al. [2020]. For a non-linear layer f_i, we compute $\mathcal{C}_{i-1}^{box}$ an over-approximation of $\mathcal{C}_{i-1}$ from Equation (2.6), satisfying:

$$\mathcal{C}_{i-1}^{box} = [\underline{x}^{(i-1)}, \overline{x}^{(i-1)}] \supseteq \mathcal{C}_{i-1} \tag{2.7}$$

Finding $\mathcal{C}_{i-1}^{box}$ requires solving the optimization problems:

$$\underline{x}^{(i-1)} = \min_{x \in \mathcal{C}} \; (\underline{A}'x + \underline{B}') \qquad \text{and} \qquad \overline{x}^{(i-1)} = \max_{x \in \mathcal{C}} \; (\overline{A}'x + \overline{B}') \tag{2.8}$$

Remark 10 (Concretization with an interval box)**.** *The operation in Equation* (2.8) *is known as concretization. When the input domain $\mathcal{C} = [\underline{x}, \overline{x}]$ is an interval box, solving this optimisation is highly efficient. It can be found independently using interval arithmetic. Using the notation $M^+ = \max(M, 0)$ and $M^- = \min(M, 0)$ (element-wise), the concrete bounds $\underline{x}^{(i-1)}$ and $\overline{x}^{(i-1)}$ are computed as:*

$$\underline{x}^{(i-1)} = \min_{x \in \mathcal{C}} \; (\underline{A}'x + \underline{B}') = \underline{A}'^{+}\underline{x} + \underline{A}'^{-}\overline{x} + \underline{B}'$$
$$\overline{x}^{(i-1)} = \max_{x \in \mathcal{C}} \; (\overline{A}'x + \overline{B}') = \overline{A}'^{+}\overline{x} + \overline{A}'^{-}\underline{x} + \overline{B}'$$

Example 7 (Detailed computation of pre-activation bounds)**.** *We illustrate the concretization step required to find the input domain of a specific layer f_i. The goal is to compute the concrete lower bound $\underline{x}^{(i-1)}$ of the interval box $\mathcal{C}_{i-1}^{box}$.*

- *Consider the coefficients $\underline{A}'$ and $\underline{B}'$ defining the linear relaxation of the sub-network preceding layer f_i (i.e., $f_{i-1} \circ \cdots \circ f_1$). Let: $\underline{A}' = \begin{pmatrix} 2 & -1 \\ 1 & 1 \\ 0 & -3 \end{pmatrix}$ and $\underline{B}' = \begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix}$.*
- *Let the initial input domain of the neural network be the interval box $\mathcal{C} = [\underline{x}, \overline{x}]$ where $\underline{x} = \begin{pmatrix} -1 \\ -2 \end{pmatrix}$ and $\overline{x} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$.*

First, we decompose $\underline{A}'$ into its positive and negative parts and apply the formula for $\underline{x}^{(i-1)}$:

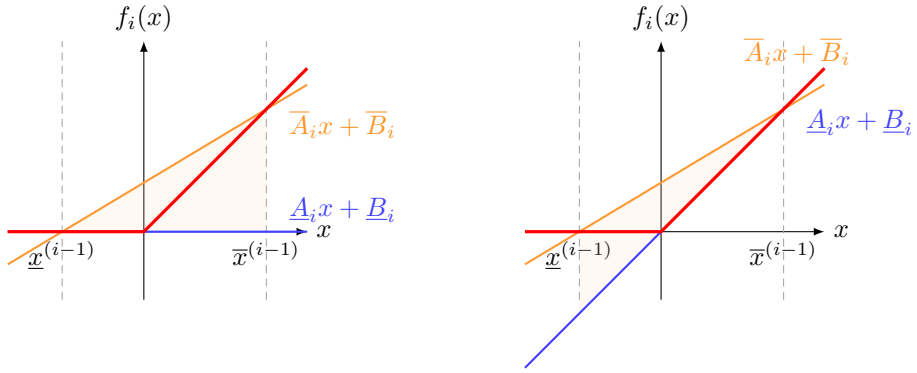
$$\underline{x}^{(i-1)} = \underline{A}'^{+}\underline{x} + \underline{A}'^{-}\overline{x} + \underline{B}' = \begin{pmatrix} 2 & 0 \\ 1 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} -1 \\ -2 \end{pmatrix} + \begin{pmatrix} 0 & -1 \\ 0 & 0 \\ 0 & -3 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix} = \begin{pmatrix} -3 \\ -3 \\ -4 \end{pmatrix}$$

2.3.3 Computing the linear relaxation coefficients per layer

Having defined the input domain $\mathcal{C}_{i-1}$, the linear relaxation coefficients $(\underline{A}_i, \overline{A}_i, \underline{B}_i, \overline{B}_i)$ introduced in Equation (2.5) must be determined. This subsection addresses the computation of these coefficients, which requires specific approximation strategies for non-linear layers. We use the ReLU function as example.

Property 2.5 (Linear relaxation of a linear layer)**.** *If $f_i(x) = A_i x + B_i$, the linear relaxation is exact on $\mathcal{C}_{i-1}$, requiring no approximation. The coefficients from Equation (2.5) are: $\underline{A}_i = \overline{A}_i = A_i$ and $\underline{B}_i = \overline{B}_i = B_i$.*

For non-linear layers, once the pre-activations bounds $\mathcal{C}_{i-1}^{box} = [\underline{x}^{(i-1)}, \overline{x}^{(i-1)}]$ from Equation (2.7) are computed, a significant amount of research, for instance recently Zhang et al. [2025], has been dedicated to optimally selecting the relaxation coefficients $(\underline{A}_i, \overline{A}_i, \underline{B}_i, \overline{B}_i)$ to achieve the tightest possible bounds. The objective is to reduce the over-approximation area of the relaxation, the orange shaded region in Figure 2.11. A smaller area corresponds to a tighter, more precise relaxation; it can be computed using Definition 2.8.



(a) Zero lower bound
$0x + 0 \leq f_i(x) \leq 0.6x + 0.7$

(b) Identity lower bound
$1x + 0 \leq f_i(x) \leq 0.6x + 0.7$

Figure 2.11: The two standard linear relaxations for non-linear ReLU layer on $\mathcal{C}_{i-1} = [\underline{x}^{(i-1)}, \overline{x}^{(i-1)}] = [-1.2, 1.8]$. Both relaxations use the same upper bound (orange) and the lower bound (blue) differ.

Definition 2.8 (Over-approximation area of a linear relaxation Shi et al. [2025])**.** *Given some pre-activations bounds $\mathcal{C}_{i-1}^{box} = [\underline{x}^{(i-1)}, \overline{x}^{(i-1)}]$ and a linear relaxation on $\mathcal{C}_{i-1}^{box}$ defined by the coefficients $(\underline{A}_i, \overline{A}_i, \underline{B}_i, \overline{B}_i)$, the area of the over-approximation region can be computed by integrating the difference between its upper and lower bounds:*

$$Area_{Linear} = \int_{\underline{x}^{(i-1)}}^{\overline{x}^{(i-1)}} \Big((\overline{A}_i x + \overline{B}_i) - (\underline{A}_i x + \underline{B}_i) \Big)\, dx$$

Example 8. *To select the tightest relaxation in Figure 2.11, we can apply Definition 2.8 to compare the over-approximation area for each relaxation (a) and (b). For the interval $[\underline{x}^{(i-1)}, \overline{x}^{(i-1)}] = [-1.2, 1.8]$:*

- $Area_{(a)} = \int_{-1.2}^{1.8} ((0.6x + 0.7) - 0)\, dx = \Big[0.6\frac{x^2}{2} + 0.7x\Big]_{-1.2}^{1.8} = 2.7$
- $Area_{(b)} = \int_{-1.2}^{1.8} ((0.6x + 0.7) - x)\, dx = \Big[-0.4\frac{x^2}{2} + 0.7x\Big]_{-1.2}^{1.8} = 1.7$

Since $Area_{(b)} < Area_{(a)}$, the "identity" relaxation (b) is tighter for this interval.

In this work, we use the widely adopted CROWN Zhang et al. [2018] and DeepPoly Singh et al. [2019] ReLU relaxation detailed in Remark 11.

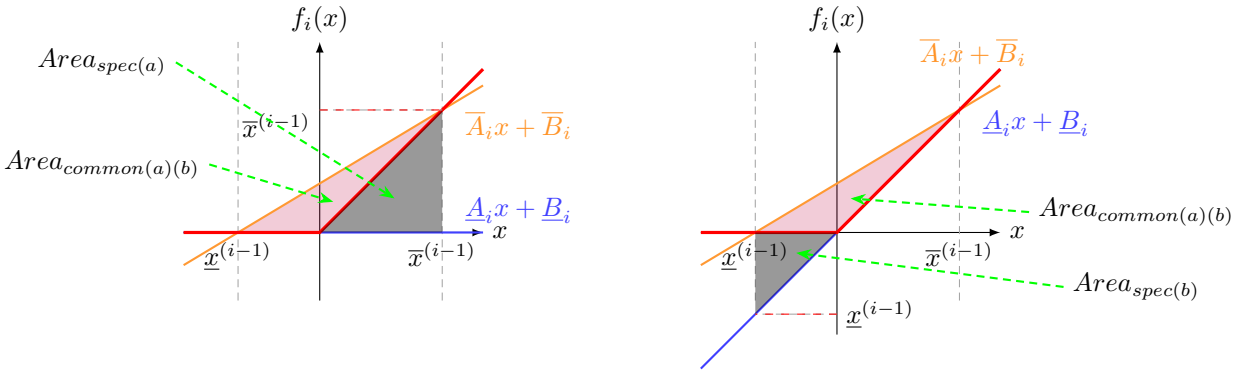
Remark 11 (Efficient ReLU Relaxation Selection)**.** *Given* $\mathcal{C}_{i-1}^{box} = [\underline{x}^{(i-1)}, \overline{x}^{(i-1)}]$*, CROWN and DeePoly methods select the tightest approximation for the ReLU between the two cases (a) and (b) in Figure 2.11. While Definition 2.8 defines tightness by the total area, a full integration is unnecessary. The total areas for both relaxations share a large common component,* $Area_{common(a)(b)}$ *(purple in Figure 2.12). The choice thus depends only on the "specific" triangular areas unique to each relaxation (black in Figure 2.12):*

- $Area_{spec(a)} = \frac{base \cdot height}{2} = \frac{(\overline{x}^{(i-1)})^2}{2}$ *(Area specific to relaxation (a))*
- $Area_{spec(b)} = \frac{base \cdot height}{2} = \frac{(\underline{x}^{(i-1)})^2}{2}$ *(Area specific to relaxation (b))*

Given $Area_{(a)} = Area_{common(a)(b)} + Area_{spec(a)}$ *and* $Area_{(b)} = Area_{common(a)(b)} + Area_{spec(b)}$*, selecting the tighter bound simplifies to comparing the specific areas:*

$$\begin{aligned} Area_{(b)} < Area_{(a)} &\iff Area_{spec(b)} < Area_{spec(a)} \\ &\iff \frac{(\underline{x}^{(i-1)})^2}{2} < \frac{(\overline{x}^{(i-1)})^2}{2} \\ &\iff |\underline{x}^{(i-1)}| < |\overline{x}^{(i-1)}| \end{aligned}$$

Since $|\underline{x}^{(i-1)}| = 1.2 < |\overline{x}^{(i-1)}| = 1.8$ *in the figure's example, the "identity" relaxation (b) is selected as it is tighter for this interval. The approach in DeepPoly and CROWN is equivalent to Definition 2.8 but computationally simpler.*



(a) Zero lower bound (b) Identity lower bound

Figure 2.12: Linear relaxations for ReLU on pre-activations bounds $\mathcal{C}_{i-1}^{box} = [\underline{x}^{(i-1)}, \overline{x}^{(i-1)}] = [-1.2, 1.8]$. Triangle in black are specific over-approximation region ($Area_{spec(a)}$ in (a) and $Area_{spec(b)}$ in (b)). The purple region ($Area_{common(a)(b)}$) is the over-approximation area common to both.

2.3.4 Composition of linear relaxations

In this subsection, we assume the layerwise relaxation coefficients $(\underline{A}_i, \overline{A}_i, \underline{B}_i, \overline{B}_i)$ under $\mathcal{C}_{i-1}$ for layer f_i and $(\underline{A}_{i+1}, \overline{A}_{i+1}, \underline{B}_{i+1}, \overline{B}_{i+1})$ for layer f_{i+1} under $\mathcal{C}_i$ are given. We show how the per-layer linear relaxations in Definition 2.7 compose pairwise to bound $f_{i+1} \circ f_i$ and, by iteration, yield bounds for the whole network $f = f_L \circ \cdots \circ f_1$.

Property 2.6 (Composition of linear relaxations Singh et al. [2019])**.** *Let* $f_i : \mathbb{R}^{k_{i-1}} \to \mathbb{R}^{k_i}$ *and* $f_{i+1} : \mathbb{R}^{k_i} \to \mathbb{R}^{k_{i+1}}$ *be two consecutive layers and* $\underline{A}_{i+1}, \overline{A}_{i+1} \in \mathbb{R}^{k_{i+1} \times k_i}$, $\underline{B}_{i+1}, \overline{B}_{i+1} \in$

$\mathbb{R}^{k_{i+1}}$ *and* $\underline{A}_i, \overline{A}_i \in \mathbb{R}^{k_i \times k_{i-1}}, \underline{B}_i, \overline{B}_i \in \mathbb{R}^{k_i}$ *the coefficients of their respective linear relaxations. We have the following inequalities:*

$$\begin{aligned} \underline{A}_{i+1}y + \underline{B}_{i+1} \;\leq\; f_{i+1}(y) \;\leq\; \overline{A}_{i+1}y + \overline{B}_{i+1} \qquad & \forall y \in \mathbb{R}^{k_i} \\ \underline{A}_i x + \underline{B}_i \;\leq\; f_i(x) \;\leq\; \overline{A}_i x + \overline{B}_i \qquad & \forall x \in \mathbb{R}^{k_{i-1}} \end{aligned}$$

First decompose the matrices $\underline{A}_{i+1}$ *and* $\overline{A}_{i+1}$ *into their positive and negative parts. For any matrix* M*, let* $M^+ = \max(M, 0)$ *and* $M^- = \min(M, 0)$ *elementwise, so* $M = M^+ + M^-$*. Starting from the relaxation for* f_{i+1} *and substituting* f_i*, we obtain:*

$$\begin{aligned} f_{i+1}(f_i(x)) \;&\geq\; \underline{A}^+_{i+1}(\underline{A}_i x + \underline{B}_i) + \underline{A}^-_{i+1}(\overline{A}_i x + \overline{B}_i) + \underline{B}_{i+1} \\ f_{i+1}(f_i(x)) \;&\leq\; \overline{A}^+_{i+1}(\overline{A}_i x + \overline{B}_i) + \overline{A}^-_{i+1}(\underline{A}_i x + \underline{B}_i) + \overline{B}_{i+1} \end{aligned}$$

Multiplying an inequality by a positive matrix preserves order, while a negative matrix reverses it coordinate-wise. The composition again has the relaxation shape:

$$\underline{A}x + \underline{B} \;\leq\; f_{i+1}(f_i(x)) \;\leq\; \overline{A}x + \overline{B} \tag{2.9}$$

$$\text{with } \underline{A} = \underline{A}^+_{i+1}\underline{A}_i + \underline{A}^-_{i+1}\overline{A}_i \in \mathbb{R}^{k_{i+1} \times k_{i-1}}, \quad \underline{B} = \underline{A}^+_{i+1}\underline{B}_i + \underline{A}^-_{i+1}\overline{B}_i + \underline{B}_{i+1} \in \mathbb{R}^{k_{i+1}},$$
$$\overline{A} = \overline{A}^+_{i+1}\overline{A}_i + \overline{A}^-_{i+1}\underline{A}_i \in \mathbb{R}^{k_{i+1} \times k_{i-1}}, \quad \overline{B} = \overline{A}^+_{i+1}\overline{B}_i + \overline{A}^-_{i+1}\underline{B}_i + \overline{B}_{i+1} \in \mathbb{R}^{k_{i+1}}.$$

The resulting relaxation in Equation (2.9) *can then be combined with the previous layer's relaxation* f_{i-1}*. This composition rule is applied iteratively through all layers of the network* $f(x) = f_L \circ f_{L-1} \circ \cdots \circ f_1(x)$*. We obtain a final linear relaxation for the entire network, with* $\underline{A}, \overline{A} \in \mathbb{R}^{k_L \times k_0}$ *and* $\underline{B}, \overline{B} \in \mathbb{R}^{k_L}$ *such that* $\forall x \in \mathcal{C}$*,* $\underline{A}x + \underline{B} \leq f(x) \leq \overline{A}x + \overline{B}$*.*

Property 2.7 (Backward bound propagation)**.** *Assuming that the pre-activation bounds* $\mathcal{C}^{box}_{i-1}$ *and the linear relaxation coefficients* $(\underline{A}_i, \overline{A}_i, \underline{B}_i, \overline{B}_i)$ *are computed for every layer* f_i*, the neural network affine bounds* $f^a(\mathcal{C})$ *are obtained by applying the composition rule iteratively through all layers of the network. The process begins with the relaxation of the final layer and successively composes it with each preceding layer, moving backward toward the input. For this reason, the method is known as a* ***backward bound propagation****, as illustrated in Figure 2.13.*

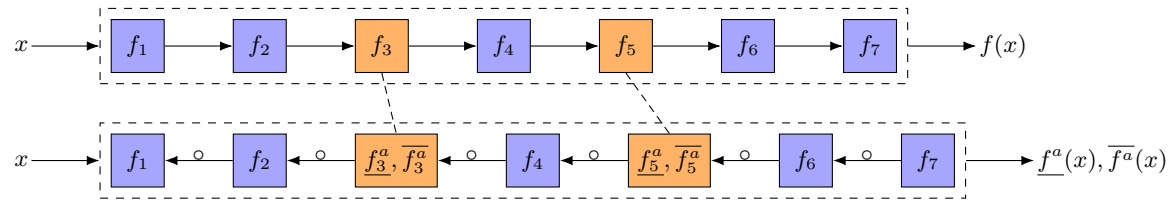


Figure 2.13: A seven-layer neural network (top) and its corresponding linear relaxation (bottom). The original network contains linear (blue) and non-linear (orange) layers. In the relaxation, the non-linear layers are replaced by linear approximations, as shown by the dashed lines. The arrows with $\circ$ illustrate the backward composition process, where f^a is constructed by composing layers sequentially from last f_7 to first f_1. The linear layers are kept as such.

Remark 12 (Associativity and computational efficiency)**.** *Since the composition of linear relaxations is associative, the grouping of layers does not alter the final result. Mathematically, the standard* ***backward propagation****:*

$$f_6 \circ (f_5 \circ (f_4 \circ (f_3 \circ (f_2 \circ f_1))))$$

yields the exact same affine bounds as a block-wise composition, such as:

$$(f_6 \circ f_5) \circ (f_4 \circ f_3) \circ (f_2 \circ f_1)$$

However, the computational cost depends heavily on the dimensions of the intermediate matrices. Let k_{hidden} be the typical width of intermediate layers (usually large) and k_L be the size of the network output. Typically, k_L is much smaller than k_{hidden} (e.g., for MNIST: $k_L = 10$ classes for classification versus $k_{hidden} \approx 784$ neurons).

- ***Block composition*** *involves multiplying coefficients between hidden layers. For example, composing $f^a_{i+1} \circ f^a_i$ results in a matrix in $\mathbb{R}^{k_{i+1} \times k_{i-1}}$. Since both dimensions are essentially k_{hidden}, these operations involve large matrices ($\approx k_{hidden} \times k_{hidden}$).*
- ***Backward propagation*** *starts from the final layer. We maintain a running relaxation matrix that maps the input of the current layer to the final network output. Consequently, the matrix has dimensions $\mathbb{R}^{k_L \times k_{i-1}}$.*

By keeping the one of the dimension fixed to the small output size k_L, the backward strategy avoids expensive matrix multiplications and is more efficient.

The composition mechanism described above provides the general framework for propagating linear bounds through the network. However, applying this framework requires specific strategies to efficiently compute the pre-activation bounds $\mathcal{C}^{box}_{i-1}$ and the relaxation coefficients for non-linear layers. The following subsections detail three standard algorithms that implement this procedure with varying trade-offs between computational cost and precision:

- Interval Bound Propagation (IBP), which simplifies the problem using constant bounds;
- CROWN, which fully leverages the precision of linear relaxations; and
- CROWN-IBP, a hybrid approach balancing speed and accuracy.

2.3.5 Special case: constant relaxation IBP

Instead of linear relaxation, we now consider a simpler method: constant relaxation. This approach, also known as **Interval Bound Propagation (IBP)**, bounds the function's output with constant values. This is a special case of the linear relaxation from Definition 2.7 where the matrices are set to zero: $\underline{A}_i = \overline{A}_i = 0$. The bounds are then defined solely by the constant bias vectors and are composed in a *forward* pass, as illustrated in Figure 2.14 and defined in Definition 2.9.

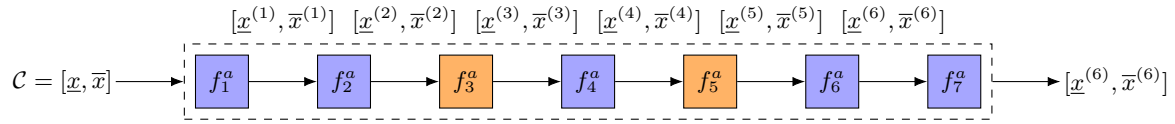


Figure 2.14: Illustration of IBP. Each layer f^a_i takes the interval from the previous layer and computes a new output interval.

Definition 2.9 (Interval Bound Propagation (IBP))**.** *Starting from the input box $\mathcal{C} = [\underline{x}^{(0)}, \overline{x}^{(0)}] = [\underline{x}, \overline{x}]$, each layer produces an interval $[\underline{x}^{(i)}, \overline{x}^{(i)}] = f^a_i([\underline{x}^{(i-1)}, \overline{x}^{(i-1)}])$, where f^a_i is generally obtained by interval arithmetic rules for that layer; some rules are given in Table 2.15.*

Name	Notation	Formula
Addition	$+$	$[\underline{a},\overline{a}]+[\underline{b},\overline{b}]=[\underline{a}+\underline{b},\overline{a}+\overline{b}]$
Substraction	-	$[\underline{a},\overline{a}]-[\underline{b},\overline{b}]=[\underline{a}-\overline{b},\overline{a}-\underline{b}]$
Positive multiplication	$\times_{\geq 0}$	$[\underline{a},\overline{a}]\cdot[\underline{b},\overline{b}]=[\underline{a}\cdot\underline{b},\overline{a}\cdot\overline{b}]$
Division	$/_{\neq 0}$	$\frac{1}{[\underline{a},\overline{a}]}=[\frac{1}{\overline{a}},\frac{1}{\underline{a}}]$
Maximum	max	$\max([\underline{a},\overline{a}],[\underline{b},\overline{b}])=[\max(\underline{a},\underline{b}),\max(\overline{a},\overline{b})]$
Minimum	min	$\min([\underline{a},\overline{a}],[\underline{b},\overline{b}])=[\min(\underline{a},\underline{b}),\min(\overline{a},\overline{b})]$

Figure 2.15: Some interval arithmetic operations.

Remark 13 (IBP is forward only)**.** *Linear relaxations retain an explicit affine dependence on the input, each layer is bounded by* $\underline{f}_i^a(x)=\underline{A}_i x+\underline{B}_i$ *and* $\overline{f}_i^a(x)=\overline{A}_i x+\overline{B}_i$*. This structure permits back-substitution through layers. By contrast, IBP represents each layer only by an interval, discarding dependencies on the network input* x*, so there is no information to propagate backward.*

IBP (constant relaxation) is less precise than linear relaxation. Figure 2.16 illustrates this for a 1D ReLU function: the area of approximation from the constant relaxation is twice the linear one.

- The linear relaxation (dashed line):

  $\forall x\in\mathcal{C}_{i-1}^{box},\quad \underline{A_i}x+\underline{B_i}=0x+0\leq f_i(x)\leq \overline{A_i}x+\overline{B_i}=0.6x+0.7$

- The constant relaxation (full line):

  $\forall x\in\mathcal{C}_{i-1}^{box},\quad \underline{B_i}=0\leq f_i(x)\leq \overline{B_i}=1.8$

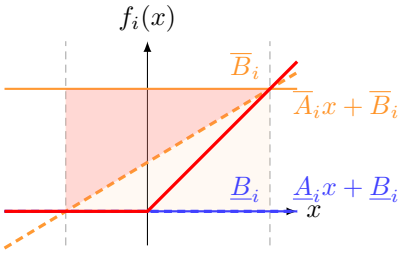


Figure 2.16: Constant and linear relaxations of a ReLU function. While both relaxations use the same lower bound (blue), their upper bounds differ. The linear relaxation's over-approximation is the orange area, while the constant relaxation's includes both the **orange and red** areas.

While IBP is computationally efficient, the loss of precision demonstrated in Figure 2.16 can be significant. To achieve tighter bounds, we return to the linear relaxation framework. The CROWN method provides a way to compute these affine bounds by leveraging the backward composition rule

2.3.6 The CROWN method

CROWN Zhang et al. [2018] is a precise and widely-used method for computing the neural network linear relaxation, that integrates the computations of pre-activation bounds and linear relaxations for each layer.

Definition 2.10 (CROWN)**.** *Given a neural network* f *and an input domain* $\mathcal{C}$*, CROWN computes the final, neural network affine bounds* $\underline{f^a}$ *and* $\overline{f^a}$*. This process involves: determining the linear relaxation coefficients* $(\underline{A}_i,\overline{A}_i,\underline{B}_i,\overline{B}_i)$ *for each layer* f_i *where* $i\in\{1,\dots,L\}$*. Then, composing all layers using backward bound propagation (Property 2.7) to get a final affine relaxation.*

However, as mentioned in subsection 2.3.3, computing the layer-wise relaxation of non-linear layers requires their ***pre-activation bounds*** $\mathcal{C}_{i-1}^{box}=[\underline{x}^{(i-1)},\overline{x}^{(i-1)}]$*. These pre-activation bounds depend on the output of all preceding layers* $(f_{i-1}^a\circ\cdots\circ f_1^a)$*. CROWN resolves this dependency by building the relaxations iteratively. For* ***each*** *non-linear layer* f_i*:*

- ***Step (a)*** *CROWN performs a backward composition on the sub-network relaxation* $(f_{i-1}^a \circ \cdots \circ f_1^a)$ *(using any relaxations already computed for layers* f_j *where* $j < i$*).*
- ***Step (b)*** *CROWN concretizes the resulting affine bounds over the input domain* $\mathcal{C}$ *to find the pre-activation bounds* $\mathcal{C}_{i-1}^{box} = [\underline{x}^{(i-1)}, \overline{x}^{(i-1)}]$*.*
- ***Step (c)*** *CROWN computes the linear relaxation* $(\underline{f_i^a}, \overline{f_i^a})$ *on the interval* $\mathcal{C}_{i-1}^{box} = [\underline{x}^{(i-1)}, \overline{x}^{(i-1)}]$*, as described in subsection 2.3.3.*

The steps (a)(b)(c) are repeated for all k *non-linear layers in the network. Once all* k *non-linear layers have been relaxed, CROWN performs one final backward composition over the entire network (which is now fully described by linear bounds) to yield the bounds* $\underline{f^a}, \overline{f^a}$*. This entire procedure requires* $k+1$ *backward passes in total.*

Example 9. *This iterative process is illustrated in Figure 2.17 for a network with 2 non-linear layers* f_3 *and* f_5*, then* $k = 2$*:*

- ***Iteration 1:*** *To relax* f_3*, backward composition is applied to the sub-network* $(f_2^a \circ f_1^a)$ *(which is equal to* $(f_2 \circ f_1)$ *because these layers are linear)* ***(a)****. After concretization, this yields the pre-activation bounds* $\mathcal{C}_2^{box} = [\underline{x}^{(2)}, \overline{x}^{(2)}]$ ***(b)****, which are used to compute the relaxation* $(\underline{f_3^a}, \overline{f_3^a})$ ***(c)****.*
- ***Iteration 2:*** *To relax* f_5*, backward composition is applied to the sub-network* $(f_4 \circ f_3^a \circ f_2 \circ f_1)$ ***(a)****. After concretization, this yields the pre-activation bounds* $\mathcal{C}_4^{box} = [\underline{x}^{(4)}, \overline{x}^{(4)}]$ ***(b)****, which are then used to compute the relaxation* $(\underline{f_5^a}, \overline{f_5^a})$ ***(c)****.*
- ***Iteration 3:*** *A final backward composition is performed over the entire relaxed network* $(f_7 \circ f_6 \circ f_5^a \circ f_4 \circ f_3^a \circ f_2 \circ f_1)$ *to compute the final bounds* $(\underline{f^a}, \overline{f^a})$*.*

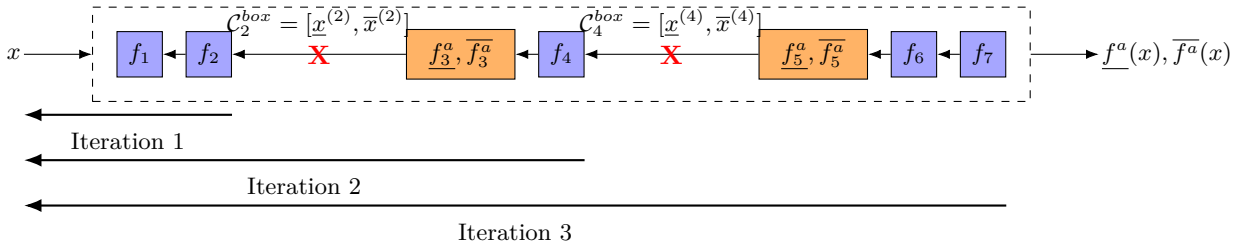
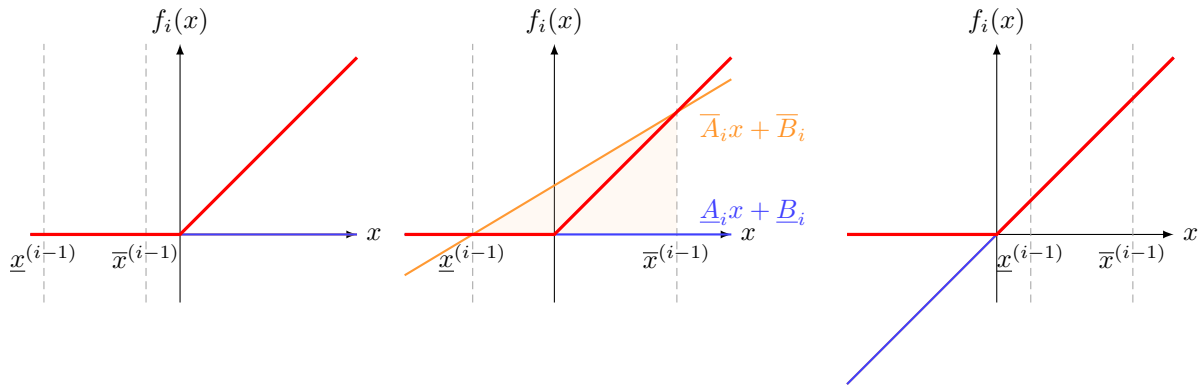


Figure 2.17: Iterative computation of pre-activation bounds in CROWN.

Remark 14 ("Backward" is shorthand)**.** *CROWN is often referred to as a* ***backward*** *method, but this is an abuse of terminology. In practice, CROWN operates in two directions: a* ***backward*** *composition of affine relaxations bounds through the network (see Property 2.7 and step (a)) and a* ***forward*** *concretization to obtain the pre-activation bounds (step (b)). Its effectiveness comes from this backward-forward coupling.*

A challenge for the CROWN method is the computational cost of the iterative procedure for computing the pre-activation bounds for non-linear layers (Definition 2.10). The following remarks discuss two approaches that use IBP to improve time efficiency.

Remark 15 (CROWN's initial IBP pass)**.** *Although* CROWN *is a backward method, it first executes a forward* IBP *pass. This pass computes interval bounds for each layer. These bounds are used to identify inactive ReLU, which are those operating in a purely linear region (cases (a) and (c) in Figure 2.18). When a layer is thus determined to be linear, its relaxation is exact, simplifying the backward pass. This does not change the precision of the final neural network affine bounds.*



(a) Inactive: $\overline{x}^{(i-1)} < 0$ (b) Active: $\underline{x}^{(i-1)} < 0 < \overline{x}^{(i-1)}$ (c) Inactive: $\underline{x}^{(i-1)} > 0$

Figure 2.18: Linear relaxation of a ReLU function for different input intervals (pre-activation) $\mathcal{C}_{i-1}^{box} = [\underline{x}^{(i-1)}, \overline{x}^{(i-1)}]$. (a)(c) An *inactive* ReLU: it stays in a linear state over the entire interval. (b) An *active* ReLU, where the interval includes the non-linear point at $x = 0$.

The iterative nature of CROWN makes it precise but computationally expensive. However, as noted in Remark 15, IBP is already employed to filter inactive neurons. This idea can be pushed further: instead of using the expensive backward procedure to find the input domain of every layer, we can rely on IBP to compute all pre-activation bounds directly. This approach leads to the CROWN-IBP variant.

Definition 2.11 (CROWN-IBP)**.** *The* CROWN-IBP *method is a variant of the CROWN method. While CROWN performs $k+1$ iterations (where k is the number of non-linear layers) as described in Definition 2.10, the CROWN-IBP performs only 2 iterations:*

- ***Iteration 1:*** *An IBP pass to estimate all the pre-activation bounds.*
- ***Iteration 2:*** *A backward bound propagation over the entire network (which is now fully described by linear bounds) to yield the bounds $\underline{f^a}, \overline{f^a}$.*

This 2-iteration process is illustrated in Figure 2.19.

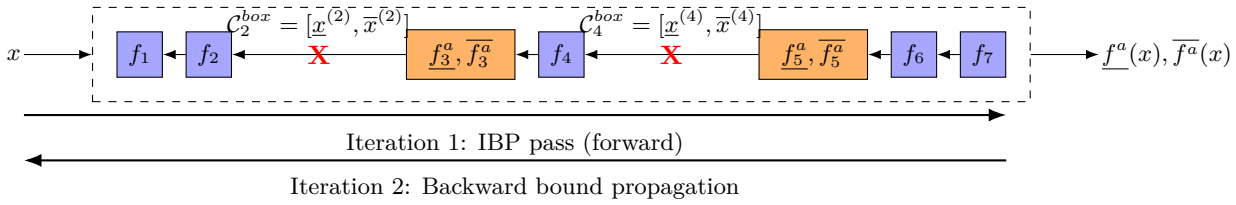


Figure 2.19: The 2-iteration process of CROWN-IBP.

Remark 16 (CROWN-IBP vs CROWN)**.** *CROWN-IBP is faster than CROWN because it uses fewer iterations. However, this speed comes at the cost of precision: relying on IBP (which is less precise than linear relaxation) can produce larger pre-activation bounds, thereby increasing the final over-approximation. In fact, the tightness of a linear relaxation is affected by the size of the pre-activation bounds, as shown in Remark 9.*

Having introduced the principles of linear and constant relaxations, we now turn to their implementation. The `auto-LiRPA` framework automates these relaxation and composition processes.

2.4 Auto-LiRPA

`auto-LiRPA` is a library implemented in PyTorch Paszke et al. [2019] to verify the reachability property. Given an input domain $\mathcal{C}$ and a neural network f implemented in PyTorch (with the supported functions[1]), `auto-LiRPA` can automatically compute $f^a(\mathcal{C})$, the (reachable) image of $\mathcal{C}$ under a relaxation of f. The relaxation can be:

- Linear[2], based on the CROWN or CROWN-IBP method, described in Subsection 2.3.6 and Definition 2.11.
- Constant, based on the IBP method, described in Subsection 2.3.5.

This section introduces the PyTorch basics and details how `auto-LiRPA` uses it to compute relaxations. This process is illustrated by an example.

2.4.1 Pytorch basics

PyTorch encodes a neural network via its computational graph, which represents the network's structure (in our case, sequential) and the functions that compose it. We begin by defining the fundamental PyTorch data type *Tensor* and then the standard Pytorch computational graph.

Definition 2.12 (Tensor Collobert et al. [2002])**.** *A tensor is an n-dimensional matrix of real-valued numbers. A property of a PyTorch tensor is its* ***shape****, which describes the size of each of its dimensions. For example, consider a picture x defined by its channels, height, and width (e.g., $3 \times 56 \times 56$) as in Definition 2.2:*

```
1    x  # Shape [3, 56, 56]
```

Listing 2.1: Example of a PyTorch tensor shape. The size of x in the 0th dimension is 3.

Definition 2.13 (PyTorch computational graph)**.** *PyTorch uses a function-based computational graph Dennis [1974]. A graph is a finite set of nodes with edges connecting pairs of nodes. In this formalism, each* ***node*** *represents a function that computes one or more output variables from one or more input variables. The variables are tensors (Definition 2.12). An* ***edge*** *from node u to node v indicates that an output variable produced by the function associated with u is consumed as an input variable by the function associated with v.*

While the full implementation of PyTorch's automatic differentiation system is complex, its high-level process consists of two distinct phases. PyTorch dynamically constructs this graph by performing a "forward pass": a standard computation running from the first function to the last. Then, from a high-level perspective Scardapane [2024], this graph is the foundation for reverse-mode automatic differentiation, which is implemented via a "backward pass": from the final function to the first. Each node has two methods within its `torch.autograd.Function` *class, as illustrated in Figure 2.20. Figure 2.21 illustrates such a computational graph with three function nodes, where the upper arrows show the forward computation from x_0 to L, and the lower arrows show the reverse-mode propagation of gradients $\frac{\partial L}{\partial x_i}$.*

[1] The following (non-exhaustive) functions are supported by `auto-LiRPA`: `Add`, `Sub`, `Mul`, `Div`, `Log`, `Sqrt`, `Reciprocal`, `Exp`, `Conv`, `ConvTranspose`, `Pad`, `Dropout`, `Cast`, `Gelu`, `Gather`, `GatherElements`, `Linear`, `MatMul`, `Neg`, `CumSum`, `Identity`, `Where`, `Not`, `Equal`, `BatchNormalization`, `MaxPool`, `GlobalAveragePool`, `AveragePool`, `Relu`, `Reshape`, `Resize`, `Expand`, `Unsqueeze`, `Squeeze`, `Flatten`, `Transpose`, `Concat`, `ConcatFromSequence`, `Slice`, `Split`, `Softmax`.

[2] Other methods also exist, such as α-CROWN Xu et al. [2020b] and β-CROWN Wang et al. [2021], but they will not be discussed in this work

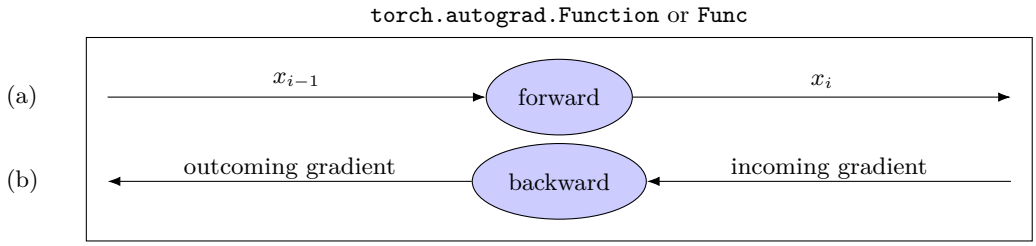


Figure 2.20: Illustration of a function node (a `Function`) in the PyTorch computational graph. (a) The `forward` method runs the standard calculation, taking tensor x_{i-1} to x_i. (b) The `bound_backward` method works in reverse for gradient calculation, taking an incoming gradient and returning an outgoing one.

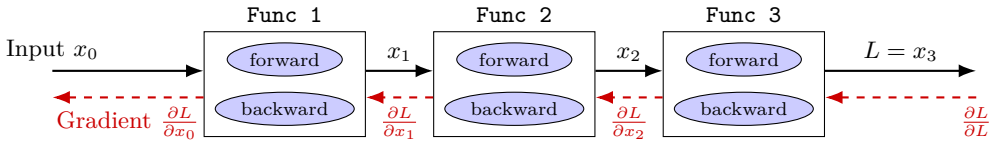


Figure 2.21: A computational graph showing the forward pass (top) for computing values and the reverse-mode automatic differentiation backward pass (bottom) for computing gradients.

Remark 17 (Link between linear relaxation and automatic differentiation)**.** *The designers of* ***auto-LiRPA*** *took inspiration from the automatic differentiation of PyTorch. Automatic differentiation can compute the gradients of a complex function automatically (based on the chain rule). The backward bound propagation in* ***auto-LiRPA*** *is analogous to reverse-mode automatic differentiation, but instead of computing gradients, it computes bounds. Both rely on computational graphs as defined in Definition 2.13.*

2.4.2 `auto-LiRPA`

`auto-LiRPA` adapts PyTorch's computational graph to automatically compute relaxations. As illustrated in Figure 2.22, `auto-LiRPA` introduces a `Bound` class specifying three methods:

- A `forward` method (a): This method must perform the standard tensor computation by calling the original PyTorch function.
- An `interval_propagate` method (b): This method must define the forward propagation of constant bounds (IBP). It must take an input interval $[\underline{x}^{(i-1)}, \overline{x}^{(i-1)}]$ and must compute an output interval $[\underline{x}^{(i)}, \overline{x}^{(i)}]$.
- A `bound_backward` method (c): This method must define the backward propagation of affine relaxations for CROWN (Definition 2.10). It must take the affine coefficients of the subsequent layers (e.g., $\underline{A}_{i+1}, \overline{A}_{i+1}$) as input and must returns the newly composed coefficients for the preceding layers (e.g., $\underline{A}_{compo}, \overline{A}_{compo}, \underline{B}_{compo}, \overline{B}_{compo}$).

  Remark 18 (Bias Accumulation)**.** *Notably, this method accepts only the slope coefficients ($\underline{A}_{i+1}, \overline{A}_{i+1}$) as input, not the bias terms ($\underline{B}_{i+1}, \overline{B}_{i+1}$). It calculates the local bias contribution of the current layer, which the framework then automatically adds to the global accumulated bias.*

Given a network f, an input domain $\mathcal{C}$ and a specified method (CROWN, CROWN-IBP or IBP), computing the bounds involves three distinct steps, as demonstrated in the code in Listing 2.2:

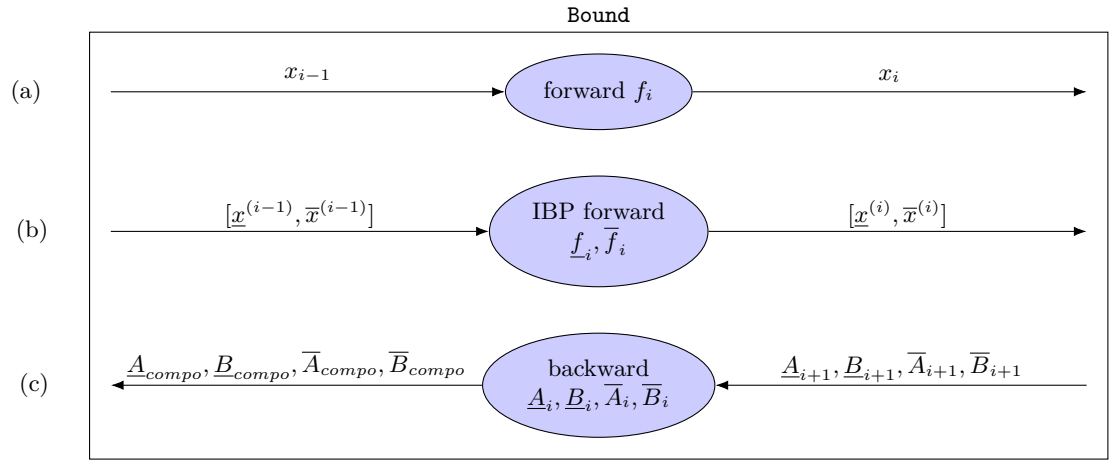


Figure 2.22: Illustration of a `Bound class` node f_i.

- **Step 1. Graph augmentation**: The neural network implemented in PyTorch is first converted into a graph called `BoundedModule`. During this initialisation, `auto-LiRPA` traces the network's functions and replaces each standard PyTorch function with its corresponding `Bound` node implementation.
- **Step 2. Domain definition**: The input perturbation $\mathcal{C}$, defined as an interval box, is encapsulated in a `BoundedTensor` object. The Listing 2.2 illustrates a similar white-noise perturbation to the one in Figure 2.4.
- **Step 3. Bound computation**: The `compute_bounds` method is called on the `BoundedModule` (Step 1) with the `BoundedTensor` (Step 2) as input. This function computes relaxation bounds based on the chosen method:
  - If IBP is selected, it executes a "forward pass" by sequentially calling the `IBP forward` method of each node.
  - If CROWN or CROWN-IBP is selected, it executes a "backward pass" by sequentially calling the `bound_backward` method of each node.

```
# Dummy PyTorch neural network
network = torch.nn.Sequential(Flatten(),
                      torch.nn.Linear(28*28, 256),
                      torch.nn.ReLU(),
                      torch.nn.Linear(256, 10))

# Step 1  Convert pytorch neural network to graph
lirpa = BoundedModule(network, torch.empty(*picture.shape))

# Step 2 Input box C: white-noise with radius 0.8 around a picture
ptb = PerturbationLpNorm(norm=float("inf"), eps=0.8)  # [-0.8, 0.8]
# picture in [0,1] from dataset
x0  = BoundedTensor(picture, ptb)  # [picture-0.8, picture+0.8]

# Step 3 Bounds on the network
lb, ub = lirpa.compute_bounds(x=(x0,), method='CROWN')
```

Listing 2.2: An `auto-LiRPA` example showing the three-step workflow. **Step 1:** The neural network is implemented in Pytorch using `torch.nn.Sequential` calling a sequence of well-known layers (e.g `torch.nn.Linear`). It is converted into a graph, the `BoundedModule`. **Step 2:** The white-noise (ℓ_∞) input perturbation domain is defined using a `BoundedTensor`. **Step 3:** The `compute_bounds` function is called to execute the chosen verification method and return the final bounds.

Remark 19 (BoundedModule and Bound)**.** ***`BoundedModule`*** *represents the entire neural network class, whereas each* ***`Bound`*** *class corresponds to a single node (operation) in the computational graph.*

2.4.3 Example: PlusConstant custom function

The `BoundedModule` handles the most commonplace PyTorch functions (e.g., `nn.Linear`). However, the neural network under study may contain other functions that are not implemented. It may also be the case that decomposing a function into known operators is not suitable for various reasons (e.g., tightness, scalability), as we describe in Chapter 3. `auto-LiRPA` offers a way to extend the graph with a *custom function* (or *custom operator*). To illustrate the full process of defining the graph and computing bounds, we now demonstrate how to implement a new function in `auto-LiRPA`. Let `PlusConstant` be a function such that: $f_i : \mathbb{R}^{k_i} \to \mathbb{R}^{k_i}$ adds a constant value $K \in \mathbb{R}^{k_i}$ (every component of K is equal to a constant k) to its input:

$$f_i(x) = x + K \tag{2.10}$$

2.4.3.1 Inserting the custom function into the computational graph

To be added to the PyTorch computational graph (Definition 2.13), the custom function needs to be defined as a `torch.autograd.Function`, as shown in Listing 2.3:

1. The `symbolic` method enables the graph insertion. It instructs the graph builder (`g`) to add a new node with a unique string identifier `'custom::PlusConstant'`.
2. The `forward` method defines the function's mathematical logic.
3. The `bound_backward` method is omitted as it is optional. This method is necessary for gradient that is used for training neural networks, but `auto-LiRPA` does not rely on it for its backward bound propagation, which is defined separately in a custom `Bound` class.

```
class PlusConstantOp(torch.autograd.Function):
    @staticmethod
    def symbolic(g, x, const):
        return g.op('custom::PlusConstant', x, const_i=const)
    @staticmethod
    def forward(ctx, x, const):
        return x + const
```

Listing 2.3: Defining a `torch.autograd.Function` to register the custom `PlusConstant` function with PyTorch's computational graph tracer.

To use this function conveniently within a neural network (e.g., `nn.Sequential`), we wrap it in a standard `torch.nn.Module`, as shown in Listing 2.4. `nn.Sequential` only accepts objects that inherit from `torch.nn.Module`, so this wrapper is required to insert `PlusConstant` into the network and handle it like any other standard layer.

```
class PlusConstant(torch.nn.Module):
    def __init__(self, const=1):
        super().__init__();
        self.const = const
    def forward(self, x):
        return PlusConstantOp.apply(x, self.const)
```

Listing 2.4: Creating a `torch.nn.Module` wrapper for `PlusConstantOp` to make it easily usable within a neural network.

2.4.3.2 Bound class

We create a subclass of `Bound`, namely `BoundPlusConstant` to implement the three methods (`forward`, `IBP forward`, `bound_backward`). The implementation is presented across the following code listings: the class definition (Listing 2.5), the forward method (Listing 2.6), the IBP method (Listing 2.7), and the backward method (Listing 2.8).

```
class BoundPlusConstant(auto_lirpa.Bound):
    def __init__(self, attr, inputs, output_index, options):
        super().__init__(attr, inputs, output_index, options)
        # attr dictionary contains the function's attributes
        self.const = attr['const'] # Retrieve the constant passed from the
            symbolic method
```

Listing 2.5: Defining the `BoundPlusConstant` class, which inherits from `auto-LiRPA`'s `Bound` class and retrieves the constant attribute.

- **Standard inference**:
  - `forward`: This method implements the mathematical function used for inference:

```
def forward(self, x):
    # standard inference
    return x + self.const
```

Listing 2.6: The `forward` method for `BoundPlusConstant`, defining the standard (non-relaxation) mathematical function.

- **Relaxation methods**:
  - `IBP forward` or `interval_propagate`: This method implements the forward constant relaxation (IBP) for this node. Given an input interval $[\underline{x}^{(i-1)}, \overline{x}^{(i-1)}]$, we must compute the output interval $[\underline{x}^{(i)}, \overline{x}^{(i)}] = f_i([\underline{x}^{(i-1)}, \overline{x}^{(i-1)}])$. Applying interval arithmetic, specifically the addition function (as shown in Table 2.15, line 1), yields:

$$[\underline{x}^{(i)}, \overline{x}^{(i)}] = f_i([\underline{x}^{(i-1)}, \overline{x}^{(i-1)}]) = [\underline{x}^{(i-1)}, \overline{x}^{(i-1)}] + K = [\underline{x}^{(i-1)} + K \ , \ \overline{x}^{(i-1)} + K]$$

```
def interval_propagate(self, *v):
    lx_previous, ux_previous = v[0]        #l for lower, u for upper
    lx = lx_previous + self.const
    ux = ux_previous + self.const
    return lx, ux
```

Listing 2.7: The `interval_propagate` method, implementing the forward constant relaxation (IBP) for the `PlusConstant` function.

  - `bound_backward`: This method implements the backward composition of affine relaxations, which is the core mechanism of CROWN. Let us denote our `PlusConstant` layer as $f_i : \mathbb{R}^{k_i} \to \mathbb{R}^{k_i}$ and the subsequent layer as $f_{i+1} : \mathbb{R}^{k_i} \to \mathbb{R}^{k_{i+1}}$. Layer f_{i+1} has the pre-activation bounds C_i and the relaxation coefficients $\underline{A}_{i+1}, \overline{A}_{i+1} \in \mathbb{R}^{k_{i+1} \times k_i}$ and $\underline{B}_{i+1}, \overline{B}_{i+1} \in \mathbb{R}^{k_{i+1}}$, the `PlusConstant` layer, $f_i(x) = x + K$, is an affine function, its relaxation is exact on $C_{i-1} \subseteq \mathbb{R}^{k_i}$. Letting $y = f_i(x)$ be the input to the subsequent layer, we have:

$$\begin{aligned} \underline{A}_{i+1} y + \underline{B}_{i+1} \ \leq \ f_{i+1}(y) \ \leq \ \overline{A}_{i+1} y + \overline{B}_{i+1} \qquad & \forall y \in C_i \\ x + K \ \leq \ f_i(x) \ \leq \ x + K \qquad & \forall x \in C_{i-1} \end{aligned}$$

Our goal is, given pre-activations bounds and linear relaxation coefficients, to find coefficients $\underline{A}_{compo}$, $\underline{B}_{compo}$, $\overline{A}_{compo}$, $\overline{B}_{compo}$ such that:

$$\underline{A}_{compo}x + \underline{B}_{compo} \;\leq\; f_{i+1}(f_i(x)) \;\leq\; \overline{A}_{compo}x + \overline{B}_{compo} \quad \forall x \in C_{i-1} \tag{2.11}$$

The composition is straightforward. We start from the inequality for f_{i+1} and substitute the exact expression for $f_i(x)$:

$$\underline{A}_{i+1}(x+K) + \underline{B}_{i+1} \;\leq\; f_{i+1}(f_i(x)) \;\leq\; \overline{A}_{i+1}(x+K) + \overline{B}_{i+1} \quad \forall x \in C_{i-1}$$

By simply rearranging the terms using the distributive property:

$$\underline{A}_{i+1}x + (\underline{A}_{i+1}K + \underline{B}_{i+1}) \;\leq\; f_{i+1}(f_i(x)) \;\leq\; \overline{A}_{i+1}x + (\overline{A}_{i+1}K + \overline{B}_{i+1}) \quad \forall x \in C_{i-1}$$

Therefore, the new affine bounds to be passed backward to the *previous* layer are:

* New weights: $\overline{A}_{compo} = \overline{A}_{i+1} \quad \in \mathbb{R}^{k_{i+1} \times k_i}$
* New bias: $\overline{B}_{compo} = \overline{A}_{i+1}K + \overline{B}_{i+1} \quad \in \mathbb{R}^{k_{i+1}}$

As explained in Remark 18, the `auto-LiRPA` framework manages the bias addition ($+\overline{B}_{i+1}$) automatically. The `bound_backward` function is only responsible for returning the additional bias component $\overline{A}_{i+1}K$.
The same logic applies to the lower bounds $\underline{A}_{compo}$ and $\underline{B}_{compo}$:

```
def bound_backward(self, lA_next, uA_next, x, *args, **kwargs):
    #l for lower, u for upper
    uA_compo= uA_next
    lA_compo= lA_next
    # self.const is K
    uB_compo= self.const*uA_next.sum(dim=list(range(2,uA_next.ndim)))
    lB_compo= self.const*lA_next.sum(dim=list(range(2,lA_next.ndim)))

    return [(lA_compo, uA_compo)], lB_compo, uB_compo
```

Listing 2.8: The `bound_backward` method, implementing the backward linear relaxation (CROWN) by composing the affine bounds.

The `sum(dim=list(range(2, A_next.ndim)))` command implements $\overline{A}_{i+1}K$, handling the multi-dimensional tensor shapes. Given that $\overline{A}_{i+1} \in \mathbb{R}^{k_{i+1} \times k_i}$ and $K \in \mathbb{R}^{k_i}$ is a vector with constant value k, the j-th component of the resulting vector $\overline{A}_{i+1}K$ is:

$$(\overline{A}_{i+1}K)_j = \sum_{l=1}^{k_i} (\overline{A}_{i+1})_{jl} K_l = \sum_{l=1}^{k_i} (\overline{A}_{i+1})_{jl} k = k \cdot \sum_{l=1}^{k_i} (\overline{A}_{i+1})_{jl}$$

2.4.3.3 Registration

Finally, we need to register the custom operator to make it visible to `auto-LiRPA`. This step creates a mapping between the unique string identifier defined in the `symbolic` method and our `BoundPlusConstant` class. This allows the `BoundedModule` to automatically instantiate the correct bound implementation when it encounters this custom operation during the graph conversion (Step 1), as shown in Listing 2.9.

```
register_custom_op("custom::PlusConstant", BoundPlusConstant)
```

Listing 2.9: Registering the `BoundPlusConstant` class.

2.4.4 More complex input domains

We have seen how `auto-LiRPA` works and how to implement a new function within it. In this thesis, as explained in the introduction, we focus on input domains. A key assumption is that `auto-LiRPA` requires the input domain $\mathcal{C}$ to be an **interval box**. Consider a neural network that takes a picture $x^\star \in \mathbb{R}^{k_0}$ as input and outputs $f(x^\star)$, as illustrated in Figure 2.23 (a). The white-noise perturbation of $x^\star$ satisfies this assumption, each pixel varies independently within $[-\epsilon, +\epsilon]$, as formalized in Example 5 and illustrated in Figure 2.23 (b).

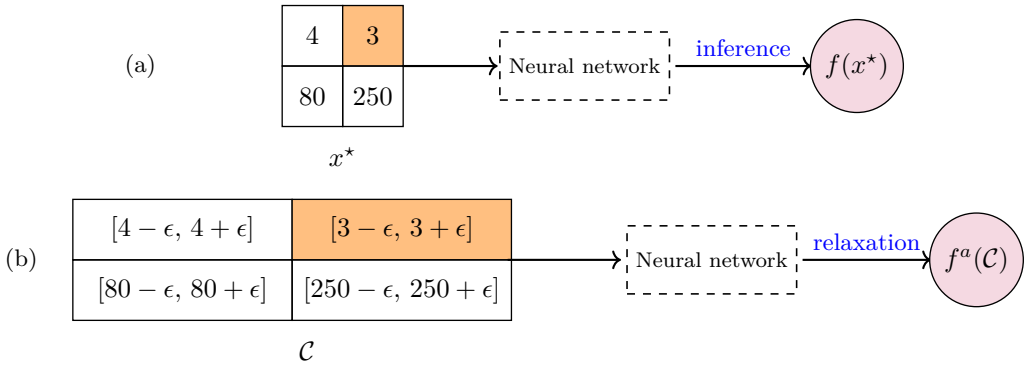


Figure 2.23: Straightforward `auto-LiRPA` usage: (a) Inference on a 4-pixels picture $x^\star$ producing $f(x^\star)$. (b) Interval box input domain $\mathcal{C}$ producing relaxed bounds $f^a(\mathcal{C})$. A specific pixel's value is highlighted in orange.

However, this assumption on the shape of the input domain is restrictive, since many important perturbations are not interval boxes in the high-dimensional pixel space. Some solutions exist; one for the brightness input domain is explained in Example 10.

Example 10 (Implementing Brightness in `auto-LiRPA` Elboher et al. [2024a])**.** *The brightness domain is a 1D line segment and not a box. As described in Example 6, the parameter of variation $\alpha_b \in [-\epsilon, +\epsilon]$ adds the same value to every pixel. Any perturbed picture $x \in \mathbb{R}^{k_0}$ in the domain can be constructed using $\alpha_b \in \mathbb{R}$ as: $x = x^\star + \alpha_b v$ where $v = (1, \ldots, 1) \in \mathbb{R}^{k_0}$. As illustrated in Figure 2.24, implementing it requires describing the perturbation $\mathcal{C}$ using the parameter of variation rather than the picture $x^\star$.*

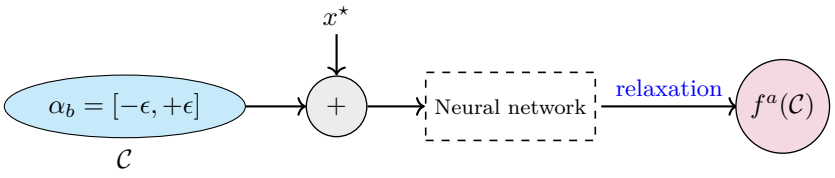


Figure 2.24: End-to-end `auto-LiRPA` pipeline for 1D brightness perturbations

*To implement it in **`auto-LiRPA`**, we can create a new first layer f_0, which takes α_b as input, $x^\star$ as an attribute and outputs the perturbed picture:*

$$f_0(\alpha_b;\ x^\star) = x^\star + \alpha_b v \qquad v = (1, \ldots, 1) \in \mathbb{R}^{k_0}$$

*This new layer is a linear transformation. This can be implemented as a standard **`torch.nn.Linear`** layer with weights $W = (1, \ldots, 1) \in \mathbb{R}^{k_0}$ and bias $b = x^\star \in \mathbb{R}^{k_0}$.*

*The original network f is then composed with this new layer f_0, creating a composite network $(f \circ f_0)$ (Figure 2.25). **`auto-LiRPA`** can then compute relaxation bounds $f^a \circ f_0^a(\mathcal{C})$ for this entire composite network.*

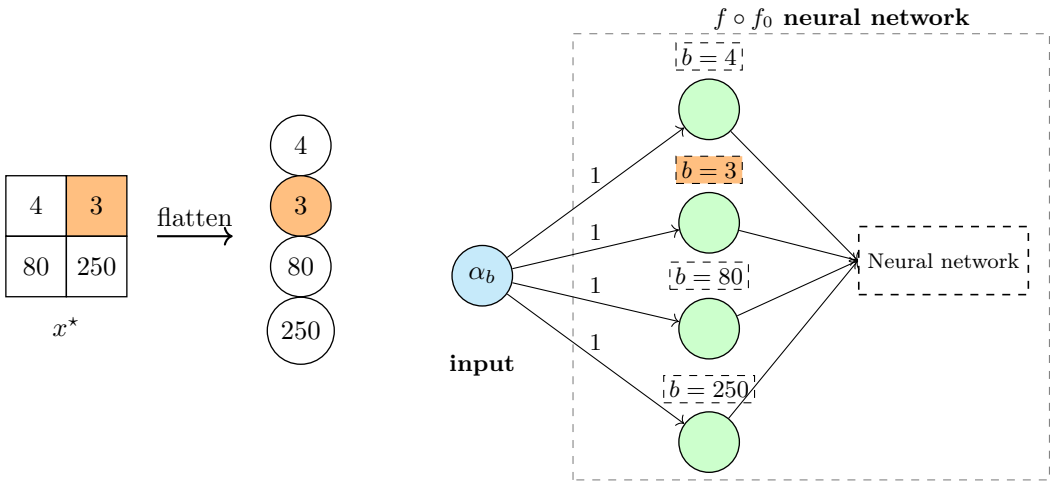


Figure 2.25: (Left) The *flatten* function converts the 2D picture matrix $x^\star$ into a 1D vector. (Right) A new linear layer (green neurons) has been added, where the flattened picture vector provides the bias terms b. This new layer takes an input α_b with connection weights of 1 and feeds its output to the original first layer of the neural network. The orange highlighting tracks a specific pixel's value.

Remark 20 (Per-picture wrapper neural network)**.** *The bias of the wrapper layer equals $x^\star$, so a separate wrapped neural network is needed for each reference input.*

Remark 21 (Encoding contrast perturbations)**.** *Contrast perturbations are also linear. They can be encoded with a wrapper layer that takes a scalar $\alpha_c \in [-\epsilon, \epsilon]$ and maps it to $x = \alpha_c\, x^\star + 0$ which is implemented as a linear layer with weights $W = x^\star$ and zero bias.*

2.5 Conclusion

In this chapter, we introduced the notion of a reachability property for reasoning about the behavior of neural networks under input perturbations. We showed how robustness to realistic perturbations, such as white noise and brightness changes, can be expressed as a particular instance of this property (Section 2.1). We defined neural networks (Section 2.2) In Section 2.3, we then concentrated on linear relaxations to approximate the reachable set $f(\mathcal{C})$. By bounding each layer with affine functions and composing these relaxations, we obtained $f^a(\mathcal{C})$. Interval Bound Propagation (IBP) corresponds to constant relaxations propagated in a forward pass (Subsection 2.3.5), whereas CROWN and CROWN-IBP leverage backward bound propagation and pre-activation bounds to obtain tighter enclosures (Subsection 2.3.6). We explain the trade-off between precision and computational complexity for these methods: tighter relaxations yield fewer *Unknown* outcomes but incur a higher computational cost. Finally, we connected these theoretical notions to an actual verification pipeline via the `auto-LiRPA` library (Section 2.4).

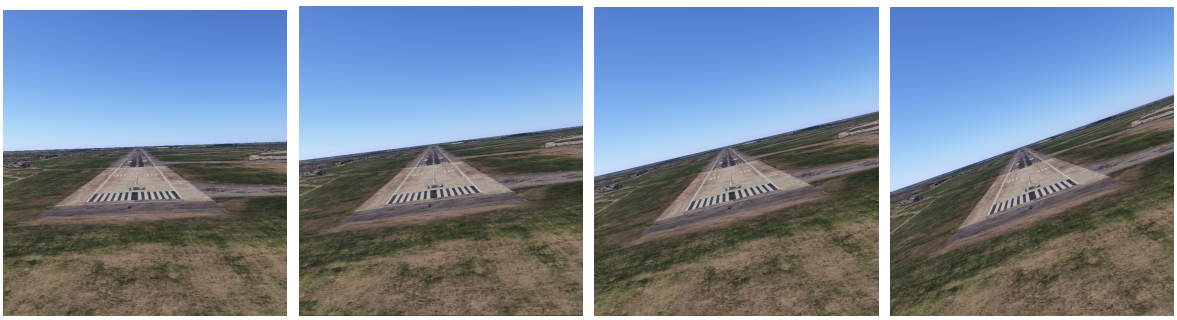
Part II

Contributions

Chapter 3

Certifying geometric robustness of neural networks

As outlined in the introduction (Chapter 1), our objective is to ensure the robustness of neural networks under geometric perturbations. An example is illustrated in Figure 3.1, where an input picture is transformed with increasing rotation angles. Specifically, we aim to verify the associated reachability property (e.g. the runway is still detected) using linear relaxation. In this chapter, the objective is to define an input domain that encompasses all pictures reachable from a reference picture through plausible geometric perturbations. Unlike white-noise and brightness perturbations introduced in the previous chapter, which modify pixel intensities while keeping pixel locations fixed, geometric transformations move pixels and interpolate their values, making the characterization of the input domain more challenging.



(a) 1° (b) 6° (c) 12° (d) 18°

Figure 3.1: 820 × 890 RGB pictures: runway viewed under increasing rotation angles.

The chapter first defines geometric transformations on pictures following Hartley and Zisserman [2003], formalizes the problem statement and reviews existing work on certifying robustness under such transformations. It finally introduces Super-DeepG, a formal verification method that verifies geometric robustness by approximating the set of perturbed images using precise linear relaxation. It employs Lipschitz optimization and GPU parallelization to achieve a trade-off between precision and time complexity.

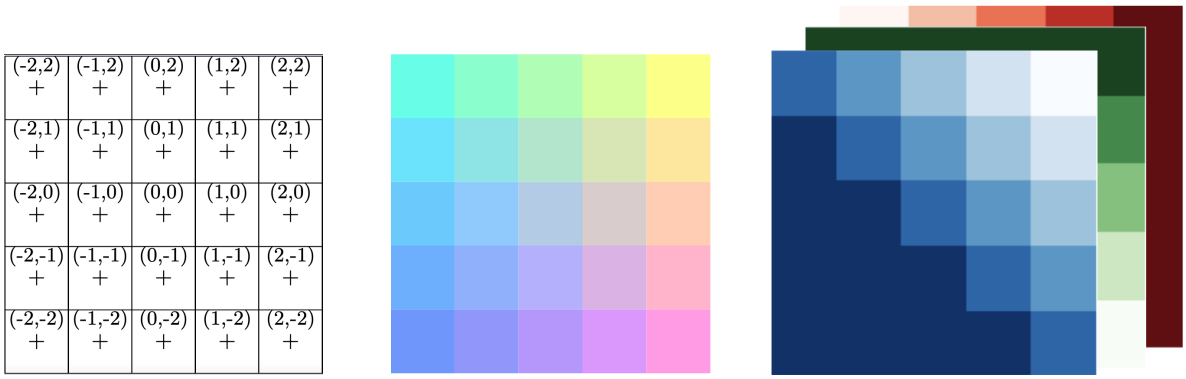
3.1 Geometric transformation

This section introduces the geometric transformations considered in this work and explains how they act on pictures. It first formalises pictures as tensors indexed by integer coordinates, then reviews the main transformation groups and their matrix representations. The section then describes how geometric transformations are implemented in practice via inverse transformation and interpolation, and concludes with a general formulation that will be used in the subsequent verification problem statement.

3.1.1 Picture

To reason about geometric robustness, it is useful to consider again a tensor-based description of a picture. A picture $x \in \mathbb{R}^{c \times n \times m}$ may be viewed as a tensor with c channels, each channel being an $n \times m$ array, where n, an odd integer, is called the width and m, an odd integer, the height. We restrict n and m to odd integers to ensure that the ranges $[-\frac{n-1}{2}, \frac{n-1}{2}]$ and $[-\frac{m-1}{2}, \frac{m-1}{2}]$ contain only integers. As a simple illustrative example, let us consider a very small 5×5-pixel picture. A picture is defined by two components:

- A grid of discrete coordinates (integer locations). We use (i, j) to denote the pixel at row i, column j. We index the grid by integer coordinates centered at 0, as illustrated in Figure 3.2 (a).
- The pixel values associated with those locations, as illustrated in Figure 3.2 (b). For this RGB[1] picture we have $c = 3$; although Figure 3.2 (b) is drawn as a two-dimensional grid, each coloured square actually represents the combination of the three channel values (R, G, B) at that coordinate (Figure 3.2 (c)). Each channel value is encoded as an integer in $[0, 255]$. For ease of expression, we write $p_{i,j}$ the pixel value at position (i, j) of a given channel.



(a) Coordinates. (b) Picture. (c) RGB channels.

Figure 3.2: A 5×5 RGB picture represented as (a) a grid of integer coordinates, (b) the resulting color image, and (c) its three separate R, G, B channels.

With this representation of coordinates and picture in place, we can now describe how geometric transformations act on them. We follow a two-step view: first, we explain how transformations act on pixel coordinates (coordinate transformation; Subsection 3.1.2), and second, how the corresponding pixel values are obtained via interpolation (picture transformation; Subsection 3.1.3).

3.1.2 Coordinate transformation

This subsection summarizes the standard hierarchy of coordinate transformations and their matrix representations in homogeneous coordinates.

Many geometric transformations exist, ranging from simple translations to full projective transformations. These transformations are organised into groups: 1) Isometries, 2) Similarities, 3) Affine transformations, 4) Projective transformations. As we move from 1) to 4), the transformations become more general, in the sense that they can model

[1] *RGB* stands for *Red*, *Green*, and *Blue*, and each pixel is represented by a triplet of intensity values (R, G, B), one for each of these three colour channels.

a wider variety of geometric distortions, at the cost of preserving geometric quantities (such as distances, angles, and parallelism).

Definition 3.1 (Homogeneous coordinates Hartley and Zisserman [2003]). *In a 2D Euclidean space, a point in two-dimensional space is denoted by Cartesian coordinates* (i, j). *Homogeneous coordinates add an extra component* $(i, j, 1)$. *This simplifies certain geometric transformations.*

In homogeneous coordinates (Definition 3.1), each transformation group is represented by a 3×3 matrix acting on pixel coordinates $(i, j, 1)$. Table 3.1 shows how these matrices transform pixel coordinates $(i, j, 1)$ into new pixel coordinates $(i', j', 1)$.

Group	**Geometric matrix**	**Transformation parameters and geometric preservation**
Isometries	$\begin{pmatrix} i' \\ j' \\ 1 \end{pmatrix} = \begin{bmatrix} \epsilon\cos\theta & -\sin\theta & t_i \\ \epsilon\sin\theta & \cos\theta & t_j \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} i \\ j \\ 1 \end{pmatrix}$	$\theta \in \mathbb{R}$ is the rotation angle, $\epsilon \in \{+1, -1\}$ encodes an optional reflection, and $t_i, t_j \in \mathbb{R}$ are translations in the i- and j-directions. Distances and angles are preserved.
Similarities	$\begin{pmatrix} i' \\ j' \\ 1 \end{pmatrix} = \begin{bmatrix} s\cos\theta & -s\sin\theta & t_i \\ s\sin\theta & s\cos\theta & t_j \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} i \\ j \\ 1 \end{pmatrix}$	$s > 0$ is a uniform scale factor, $\theta \in \mathbb{R}$ is the rotation angle, and t_i, t_j are translations. Shapes are preserved up to a global scaling of distances.
Affine	$\begin{pmatrix} i' \\ j' \\ 1 \end{pmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_i \\ a_{21} & a_{22} & t_j \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} i \\ j \\ 1 \end{pmatrix}$	The entries $a_{11}, a_{12}, a_{21}, a_{22}$ form a general 2×2 linear transformation that can encode rotations, anisotropic scalings, and shears, while t_i, t_j are translations. Parallel lines are preserved.
Projective	$\begin{pmatrix} i_a \\ j_a \\ w_a \end{pmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_i \\ a_{21} & a_{22} & t_j \\ h_1 & h_2 & h_3 \end{bmatrix} \begin{pmatrix} i \\ j \\ 1 \end{pmatrix}$, $(i', j') = \left(\frac{i_a}{w_a}, \frac{j_a}{w_a}\right)$	The additional parameters h_1, h_2, h_3 control perspective effects: straight lines remain straight, but parallel lines may intersect at vanishing points, and the local scale varies with position through the division by w_a.

Table 3.1: Matrix representations of the groups and their transformation parameters.

In this work, we restrict our attention to the transformations used in current state-of-the-art robustness verification methods, listed in Table 3.2. These transformations can be composed, but for simplicity and development time we focus on single transformations; the verification problem is essentially the same, and our code readily extends to any compositions.

Inverse rotation	**Inverse translation**	**Inverse scaling**	**Inverse shearing**
$\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & -t_i \\ 0 & 1 & -t_j \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1/\lambda & 0 & 0 \\ 0 & 1/\lambda & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & -m & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Table 3.2: Matrix representations of the geometric transformations considered in this work.

While these matrices describe how geometric transformations act on pixel coordinates,

we must still explain how to recover the corresponding pixel values.

3.1.3 Picture transformation

We now detail these geometric transformations as operators on pictures. A geometric picture transformation is usually defined as a function g that, given a picture $x \in \mathbb{R}^{c\times n\times m}$ and some transformation parameters (e.g. a rotation angle, a translation vector, or a scaling factor) denoted $\kappa \in \mathbb{R}^d$, returns the transformed picture $g(x, \kappa)$:

$$g : \begin{cases} \mathbb{R}^d \times \mathbb{R}^{c\times n\times m} \to \mathbb{R}^{c\times n\times m} \\ x, \kappa \qquad\qquad\quad \mapsto g(x, \kappa) \end{cases}$$

In practice, rather than starting from the original picture and transforming its pixels forward, it is more convenient to start from the *empty transformed picture* and *fill it in*. To do so, the approach is to:

- Apply the inverse transformation to a coordinate of the transformed picture (i, j) to find the corresponding coordinates (i', j') in the original grid.
- Read the value at (i', j') from the original picture.

For example, in Figure 3.3, for an inverse rotation $R_\theta^{-1} : \mathbb{Z}^2 \to \mathbb{R}^2$ by an angle $\theta = 70° \in \mathbb{R}$, the red asterisk at position $(2, 0)$ in the transformed grid corresponds to the point $(i', j') \approx (0.68, -1.88)$ in the original grid.

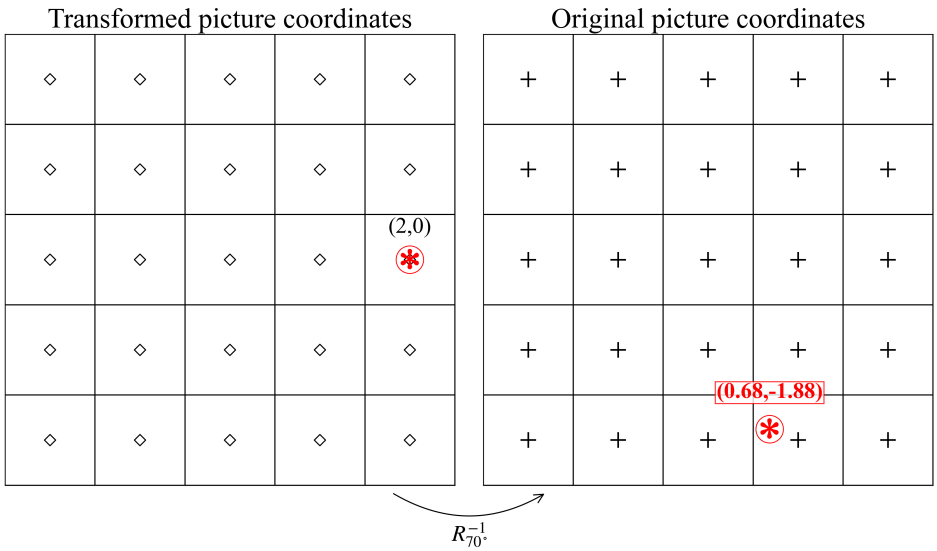


Figure 3.3: Starting from a coordinate in the transformed picture, (a) the inverse rotation $R_{70°}^{-1}$ is applied to find the corresponding location in the original coordinates.

Since (i', j') does not coincide with an integer grid point, its pixel value cannot be read directly from the original picture (as is the case in Figure 3.3). It is instead calculated from the values of a **neighborhood** of (i', j'). To illustrate this interpolation process, we decompose the process into distinct steps.

3.1.3.1 Interpolation steps: selection of neighbor pixels

Among many existing interpolation rules, the standard choice in geometric robustness Balunovic et al. [2019]; Yang et al. [2022]; Batten et al. [2024] is bilinear interpolation, which considers four neighbours. Given $(i', j') \in \mathbb{R}^2$, define the integer (i_0, j_0) coordinates:

$$i_0 = \lfloor i' \rfloor \in \mathbb{N}, \qquad j_0 = \lfloor j' \rfloor \in \mathbb{N},$$

The four integer neighbours of (i', j') are:

$$v_0 = (i_0, j_0), \qquad v_1 = (i_0, j_0 + 1),$$
$$v_2 = (i_0 + 1, j_0), \quad v_3 = (i_0 + 1, j_0 + 1),$$

with corresponding pixel values: p_0 for v_0, p_1 for v_1, p_2 for v_2, p_3 for v_3. In Figure 3.4 (a), the four neighbour coordinates are highlighted in blue: $v_0 = (0, -2)$ at the bottom left, $v_1 = (0, -1)$ at the top left, $v_2 = (1, -2)$ at the bottom right, and $v_3 = (1, -1)$ at the top right: In Figure 3.4 (b), their pixel values are retrieved.

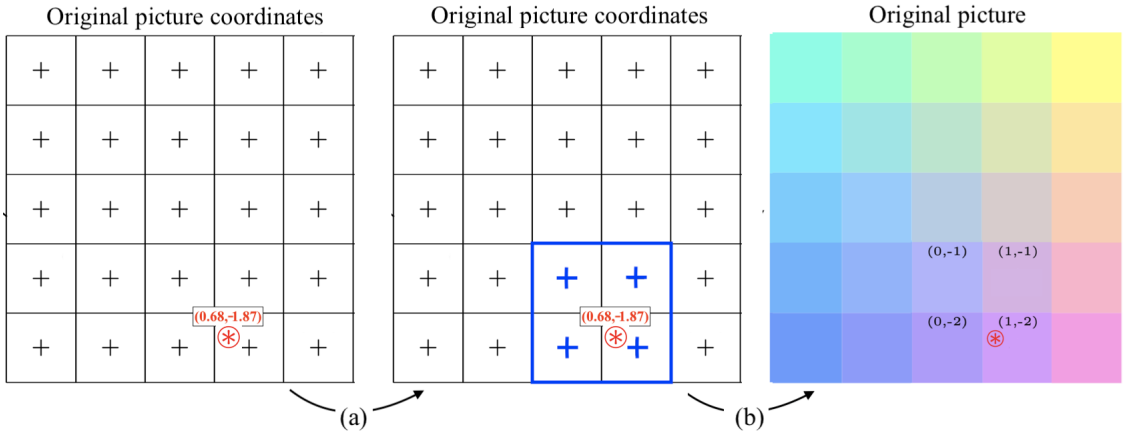


Figure 3.4: The four nearest neighbour coordinates (a) and pixel values (b) in the original picture are identified for interpolation.

3.1.3.2 Interpolation steps: Interpolation weights

Having identified the four neighboring pixels and their values, the next step is to determine how much each neighbor contributes to the value at (i', j') through suitable interpolation weights. Consider the offsets α, β between (i_0, j_0) and (i', j'):

$$\alpha = i' - i_0 \in [0, 1], \qquad \beta = j' - j_0 \in [0, 1].$$

Figure 3.5 details this local configuration: the point (i', j') lies inside the cell spanned by the integer corners v_0, v_1, v_2, v_3, and the arrows indicate the offsets α and β.

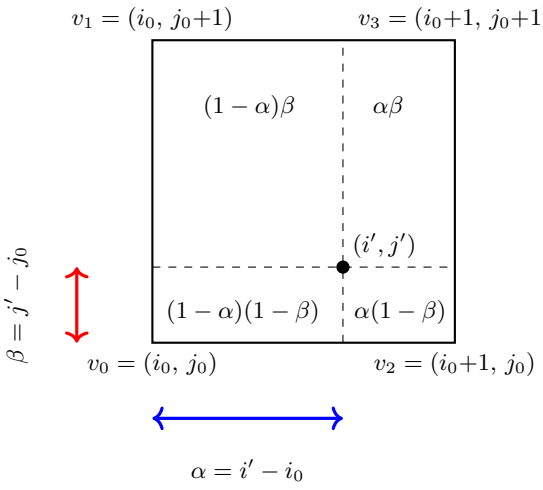


Figure 3.5: The point (i', j') is within the cell where the four corners are v_0, v_1, v_2, v_3.

The standard bilinear interpolation of x at (i', j') is given by

$$I_x(i', j') \;=\; (1-\alpha)\,(1-\beta)\,p_0 \;+\; (1-\alpha)\,\beta\,p_1 \;+\; \alpha\,(1-\beta)\,p_2 \;+\; \alpha\,\beta\,p_3 \tag{3.1}$$

Thus, the value at (i', j') is a convex combination of the four neighboring pixels. In Figure 3.6, the pixel values of the neighbors are represented by four squares. Their values are combined (Equation (3.1)) to produce the final value of the corresponding pixel in the transformed picture:

$$I_x(0.68, -1.87) = 0.28\,p_0 \;+\; 0.042\,p_1 \;+\; 0.59\,p_2 \;+\; 0.088\,p_3$$

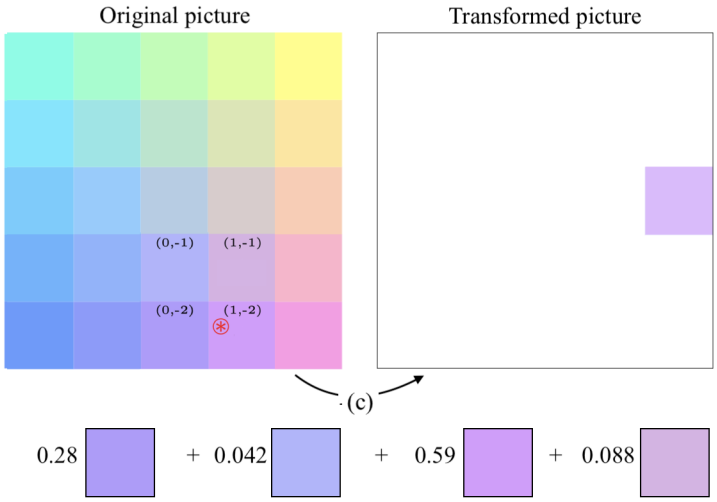


Figure 3.6: The four nearest neighbors' pixel values are combined to determine the final color of the pixel in the transformed picture.

Remark 22 ((i', j') is an integer)**.** *When (i', j') coincides with one of the grid points v_k, the interpolation reduces to the exact pixel value of the original picture at that point, i.e., $I_x(i', j') = p_k$. Table 3.3 summarizes the four possibilities.*

(i', j')	α	β	$I_x(i', j')$
v_0	0	0	p_0
v_1	0	1	p_1
v_2	1	0	p_2
v_3	1	1	p_3

Table 3.3: Bilinear interpolation in the integer case $(i', j') = v_k$.

Remark 23 (Channel-wise interpolation)**.** *Bilinear interpolation is applied independently to each channel.*

Remark 24 (Boundary pixels)**.** *When (i', j') lies close to the boundary of the picture, some of its four neighbors v_0, v_1, v_2, v_3 may fall outside the grid of integer coordinates. In that case, the interpolation rule must be combined with a padding convention. In our setting, we adopt black (zero) padding: any neighbor v_i outside the picture is treated as having pixel value $p_i = 0$. Other padding schemes (e.g. border replication or reflection) could be incorporated easily in our implementation, but we follow the standard choice in geometric robustness and use black padding.*

Transformation and interpolation have been presented for a single output pixel and a specific rotation, but the same mechanism extends to any transformations and to all pixels simultaneously.

3.1.4 General formulation of geometric transformations

We now reformulate how geometric transformations act in the general case. Let T_κ^{-1} denote any inverse geometric transformation (or composition). We denote by $p_{i,j}(\kappa)$ the pixel value at position (i, j) in the transformed picture $g(x, \kappa)$ (of an arbitrary channel). The value associated with this pixel is defined by:

$$p_{i,j}(\kappa) = I_x \circ T_\kappa^{-1}(i, j) \tag{3.2}$$

where I_x is an interpolation function. Note that $p_{i,j}(0)$ denotes the value of the original picture x at (i, j).

Some examples of geometric transformations applied to the picture (already seen in Figure 3.2) are shown in Figure 3.7.

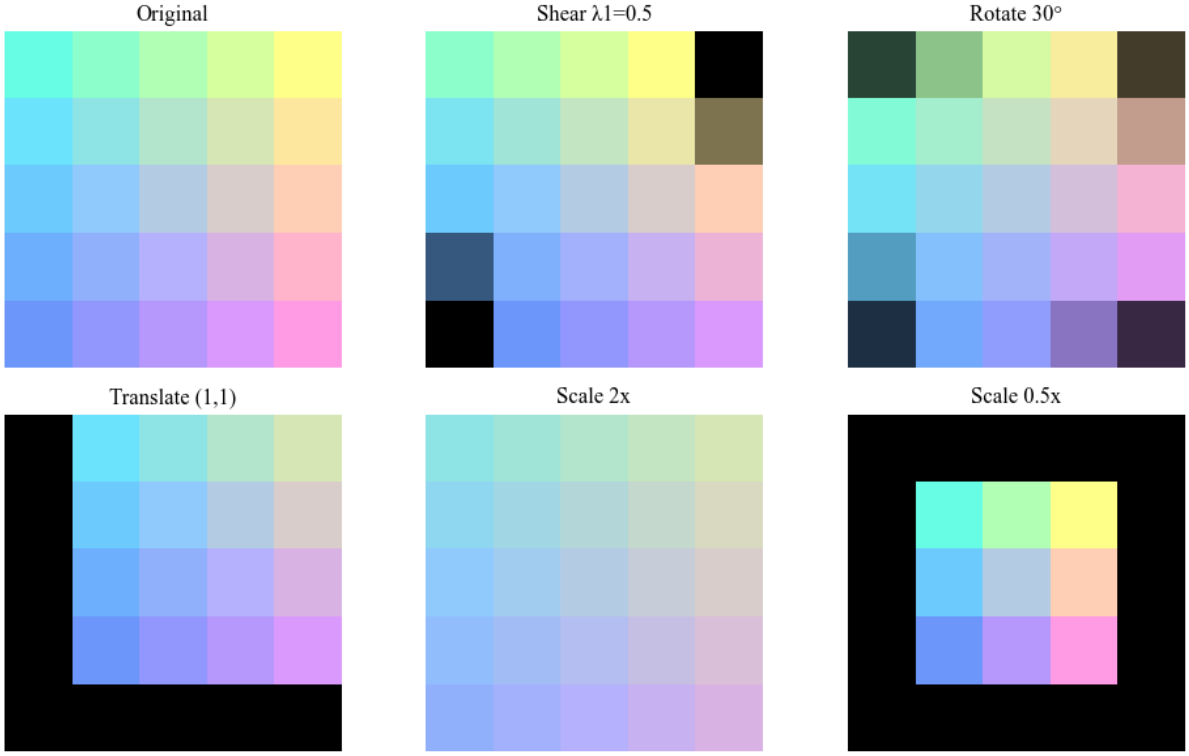


Figure 3.7: Illustration of geometric transformations applied to a 5×5 RGB grid picture. The first panel shows the original picture, while the others display the effects of shearing, rotation, translation, and scaling. All transformed pictures preserve the original resolution and appear smoothly interpolated.

Remark 25 (Black interpolated pixels)**.** *Figures 3.7 and 3.1 (runway rotations) highlight an important distinction between synthetic and physical rotations. In Figure 3.7, a geometric transformation is applied to a finite, fixed-size picture. The new pixel values are obtained by interpolating the original ones, but some output coordinates are mapped outside the size of the original picture; these undefined pixels are then filled with a default background color, which appears as black (see Remark 24). By contrast, in Figure 3.1, each rotated view is a fresh acquisition of the scene: the camera itself has rotated, so every pixel corresponds to an actually observed location and no artificial black padding is introduced.*

Having clarified how geometric transformations act on coordinates and pixel values, we now turn to their implementation in Pytorch.

3.1.5 Implementation of geometric transformations in PyTorch

Geometric transformations can be applied with minimal implementation effort in PyTorch. The computation follows the same idea as in Equation (3.2): for each output pixel, an inverse transformation gives a coordinate in the original picture, and bilinear interpolation provides its value.

The procedure, illustrated in Listings 3.1–3.4, can be decomposed into three steps:

- **Step A: build the transformation matrix.** From the transformation parameters (e.g. a rotation angle), construct a 2×3 affine matrix representing the inverse transformation T_κ^{-1} that acts on coordinates. This matrix corresponds to the top two rows of the homogeneous 3×3 matrix in Table 3.1.

```python
import torch, math
import torch.nn.functional as F

x = ... # x input picture tensor has shape (batch, 3, n, m)

def rotate_image(x, theta_degrees):
    # Step A: build the 2x3 affine matrix from the angle
    theta = math.radians(theta_degrees)
    cos_t, sin_t = math.cos(theta), math.sin(theta)

    # # Shape (1, 2, 3): batch size 1, 2x3 affine matrix
    inv_tranf_rot = torch.tensor(
        [[[cos_t, sin_t, 0.0],
          [-sin_t,  cos_t, 0.0]]], dtype=torch.float32).to(x.device)
```

Listing 3.1: PyTorch implementation of step A illustrated with a rotation.

- **Step B: build the coordinate grid.** Use `torch.nn.functional.affine_grid` to convert the affine matrix into a grid of source coordinates. For each output pixel, this grid stores the (possibly non-integer) coordinate (i', j') in the original picture and therefore implements the discrete inverse transform T_κ^{-1}.

```python
    # The grid is in normalised coordinates in [-1, 1]^2.
    # size=x.size() ensures the output has the same resolution as x.
    grid = F.affine_grid(inv_tranf_rot, size=x.size())
```

Listing 3.2: PyTorch implementation of step B: build the coordinate grid.

- **Step C: interpolate the picture.** Call `torch.nn.functional.grid_sample` with the original picture and the coordinate grid. For every grid point, `grid_sample` reads the pixel value in the original picture using bilinear interpolation, and returns the transformed picture.

```python
    x_rot = F.grid_sample(x, grid, mode="bilinear", padding_mode="zeros")
    return x_rot
```

Listing 3.3: PyTorch implementation of the step C: interpolate the picture.

A usage example is shown below.

```python
x_rotated = rotate_image(x, theta_degrees=30.0)
```

Listing 3.4: Example usage of Steps A,B,C illustrated with a rotation.

By replacing the `inv_tranf_rot` tensor with another transformation tensor, one obtains the inverse of translations, scalings, shears, and combinations thereof, following the matrices in Table 3.1.

In summary, we now have a precise mathematical description of geometric transformations on picture. We next turn to how to verify geometric robustness.

3.2 Geometric robustness: Problem statement

In this section, we formalize the verification problem induced by geometric perturbations. The analysis builds upon the interval and linear relaxations introduced in the previous chapter (Section 2.3).

In the context of robustness verification, we analyze neural networks around a reference picture x by studying the variations caused by geometric transformations. We can express the perturbed picture as a function of the transformation parameters κ:

$$g_x : \begin{cases} \mathbb{R}^d \to \mathbb{R}^{c \times n \times m} \\ \kappa \mapsto g_x(\kappa) \end{cases}$$

where $g_x(\kappa)$ denotes the transformed version of the fixed picture x under parameters κ and transformation g.

Our goal is to define an *input domain*, denoted $g_x^{[\underline{\kappa},\overline{\kappa}]}$, that encompasses all pictures that can be reached from the reference picture x when the transformation parameters vary within the range of perturbations $[\underline{\kappa},\overline{\kappa}] \subseteq \mathbb{R}^d$:

$$\textit{The set of transformed pictures:} \qquad g_x^{[\underline{\kappa},\overline{\kappa}]} = \{g_x(\kappa) \mid \kappa \in [\underline{\kappa},\overline{\kappa}]\} \tag{3.3}$$

We first describe $g_x^{[\underline{\kappa},\overline{\kappa}]}$, then use relaxation techniques to obtain an over-approximation.

3.2.1 Characterising the set of transformed pictures

To characterize $g_x^{[\underline{\kappa},\overline{\kappa}]}$, we first examine a single pixel transformation and then explain why an exact description of $g_x^{[\underline{\kappa},\overline{\kappa}]}$ is difficult, if not impossible to obtain. We illustrate this with an example of an inverse rotation, showing how the range of perturbations $[\underline{\kappa},\overline{\kappa}]$ acts on pixel coordinates and then how the resulting pixel values are computed by interpolation.

Under such perturbations, the coordinates of a pixel follow a specific trajectory in the coordinate space. As illustrated in Figure 3.8, considering an inverse rotation by an angle $\theta \in \mathbb{R}$ over the range of perturbation $[\underline{\theta},\overline{\theta}] = [0°, 70°]$, we observe that a specific pixel coordinate at $(2, 0)$ continuously traces a red path from its origin at $(2, 0)$, sweeping through all intermediate positions until it arrives at $(0.68, -1.88)$.

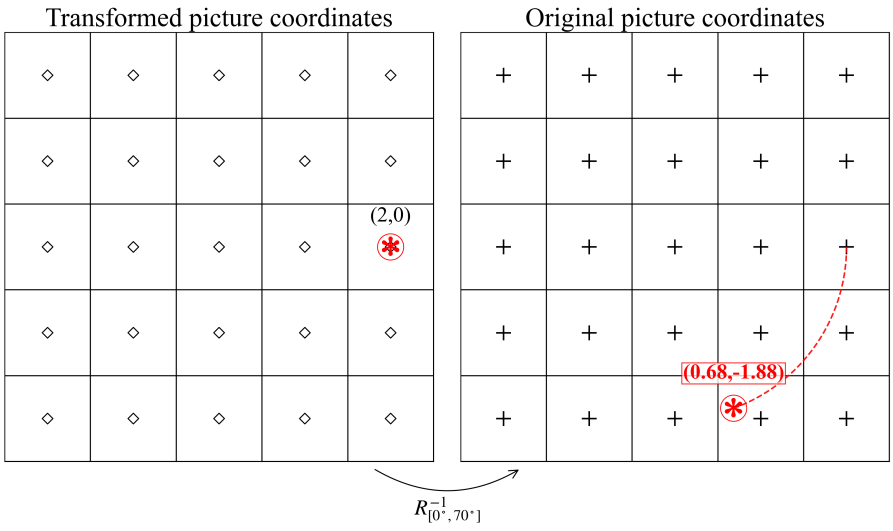
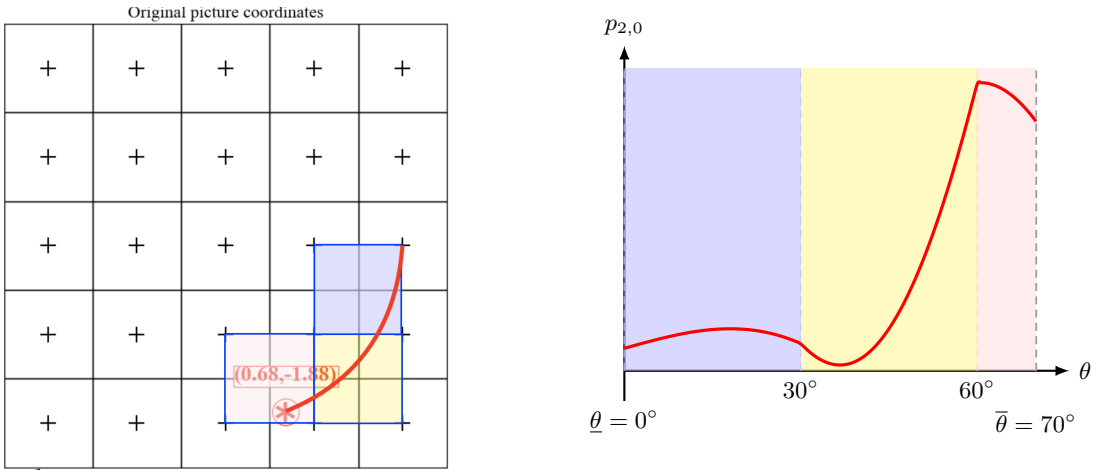


Figure 3.8: The inverse rotation R^{-1} over $[0°, 70°]$ traces a trajectory in the original picture coordinates.

Once these transformed coordinates are obtained, the corresponding pixel values must be determined through interpolation, for instance using bilinear interpolation introduced earlier in Equation (3.1). Finally, the pixel transformation for the inverse rotation at coordinate (i, j), $p_{i,j}(\theta) = I_x \circ R_\theta^{-1}(i, j)$ can be expressed as:

$$p_{i,j}(\theta) = \sum_{k=-n/2}^{n/2} \sum_{l=-m/2}^{m/2} p_{k,l}(0) \cdot \max(0, 1 - |(i\cos(\theta) + j\sin(\theta)) - k|) \cdot \max(0, 1 - |(j\cos(\theta) - i\sin(\theta)) - l|) \tag{3.4}$$

This interpolation formulation of Equation (3.4) is mathematically equivalent to the *local* definition given in Equation (3.1), but instead sums over all pixels in the grid, where the weights for non-neighboring pixels evaluate to zero. Equation (3.4) is non-linear and non-polynomial with respect to θ. As illustrated in Figure 3.9 (a), the coordinate trajectory (in red) crosses different interpolation neighborhoods (purple, yellow, pink). The resulting pixel value oscillates multiple times, as in Figure 3.9 (b). The function $p_{2,0}(\theta)$ is non-convex.



(a) Coordinate view

(b) Pixel view

Figure 3.9: (a) In the original coordinates, the inverse rotation R^{-1} over $\theta \in [0°, 70°]$ crosses three interpolation neighborhoods (purple, yellow, pink). (b) Corresponding pixel value $p_{2,0}$.

In general, each function $p_{i,j}(\kappa)$ is non-linear, non-polynomial and generally non-convex on the range of perturbations $[\underline{\kappa}, \overline{\kappa}]$.

Proposition 1. *Each function $p_{i,j}(\kappa)$ is continuous.*

Proof 1 (Proof of Continuity)**.** *For a fixed output pixel (i, j), the value $p_{i,j}(\kappa) = I_x \circ T_\kappa^{-1}(i, j)$ is a composition of continuous operations. The inverse transformation T_κ^{-1} is continuous in κ, and the interpolation function I_x involves only absolute values and the maximum function, both continuous. Since finite sums, products, and compositions of continuous functions remain continuous, $p_{i,j}(\kappa)$ is continuous with respect to κ.*

Furthermore, the $n \times m$ pixels constituting the tensor $g_x^{[\underline{\kappa}, \overline{\kappa}]}$ vary in distinct, independent ways, as illustrated in Figure 3.10. Consequently, determining the exact shape of $g_x^{[\underline{\kappa}, \overline{\kappa}]}$ in the high-dimensional space $\mathbb{R}^{c \times n \times m}$ is a highly complex task. This motivates the use of relaxation techniques.

$$g_x(\theta) = \begin{pmatrix} p_{-\frac{n}{2},\frac{m}{2}}(\theta) & \dots & p_{\frac{n}{2},\frac{m}{2}}(\theta) \\ \vdots & \ddots & \vdots \\ p_{-\frac{n}{2},-\frac{m}{2}}(\theta) & \dots & p_{\frac{n}{2},-\frac{m}{2}}(\theta) \end{pmatrix}$$

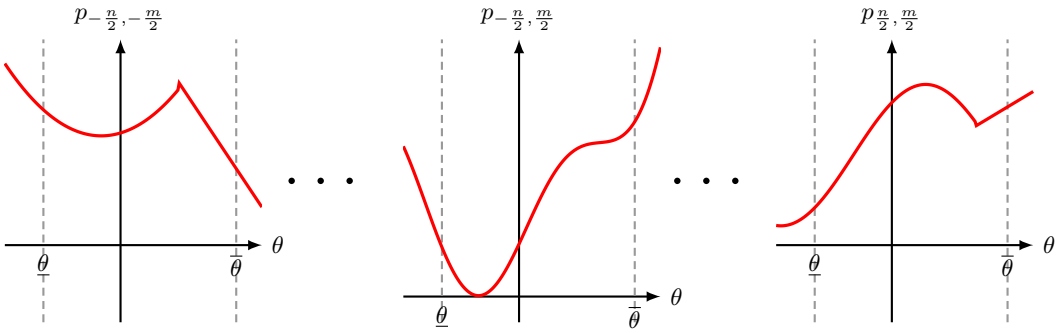


Figure 3.10: Schematic illustration highlighting the heterogeneity of pixel variations for 3 pixels $p_{-\frac{n}{2},-\frac{m}{2}}$, $p_{-\frac{n}{2},\frac{m}{2}}$, $p_{\frac{n}{2},\frac{m}{2}}$.

3.2.2 Relaxation of the set of transformed pictures

To make the verification tractable, we construct an over-approximation of the set of transformed pictures $g_x^{[\underline{\kappa},\overline{\kappa}]}$ using relaxation techniques. To achieve this, we compute a fast, tight relaxation of the geometric transformation g_x under the range of perturbations $[\underline{\kappa},\overline{\kappa}]$. We seek the linear relaxation coefficients $\underline{A},\overline{A} \in \mathbb{R}^{c\times n\times m\times d}$ and $\underline{B},\overline{B} \in \mathbb{R}^{c\times n\times m}$ such that:

$$\forall \kappa \in [\underline{\kappa},\overline{\kappa}], \qquad \underline{A}\cdot\kappa + \underline{B} \;\leq\; g_x(\kappa) \;\leq\; \overline{A}\cdot\kappa + \overline{B}. \tag{3.5}$$

IBP (constant bounds) corresponds to the special case where $\underline{A} = \overline{A} = 0$. Using linear or constant relaxation, we denote this over-approximation of the transformed picture set as $g_x^{a[\underline{\kappa},\overline{\kappa}]}$, which satisfies under the range of perturbations $[\underline{\kappa},\overline{\kappa}]$:

$$g_x^{[\underline{\kappa},\overline{\kappa}]} \subseteq g_x^{a[\underline{\kappa},\overline{\kappa}]} \tag{3.6}$$

This relaxation is computed independently per pixel $p_{i,j}$ under the range of perturbations $[\underline{\kappa},\overline{\kappa}]$ with coefficient $\underline{a},\overline{a} \in \mathbb{R}^d$ and bias tensors $\underline{b},\overline{b} \in \mathbb{R}$ such that:

$$\forall \kappa \in [\underline{\kappa},\overline{\kappa}], \qquad \underline{a}\cdot\kappa + \underline{b} \;\leq\; p_{i,j}(\kappa) \;\leq\; \overline{a}\cdot\kappa + \overline{b}. \tag{3.7}$$

Similarly, IBP can be viewed as a special case where $\underline{a} = \overline{a} = 0$. In this case, we denote the constant bounds as $\underline{p_{i,j}}$ and $\overline{p_{i,j}}$:

$$\forall \kappa \in [\underline{\kappa},\overline{\kappa}], \qquad \underline{p_{i,j}} \;\leq\; p_{i,j}(\kappa) \;\leq\; \overline{p_{i,j}} \tag{3.8}$$

We highlight four main aspects of this relaxation:

1. **Pixel-wise independence.** These bounds are constructed independently for each pixel, as illustrated for linear bounds in Figure 3.11 and for constant bounds in Figure 3.12.

2. **Integration with the neural network.** Together, these bounds per-pixel define the input domain for the verification of the neural network. They can be composed with the subsequent layers of the neural network using the CROWN/DeepPoly or IBP methods described in the previous chapter and the verification can be done.

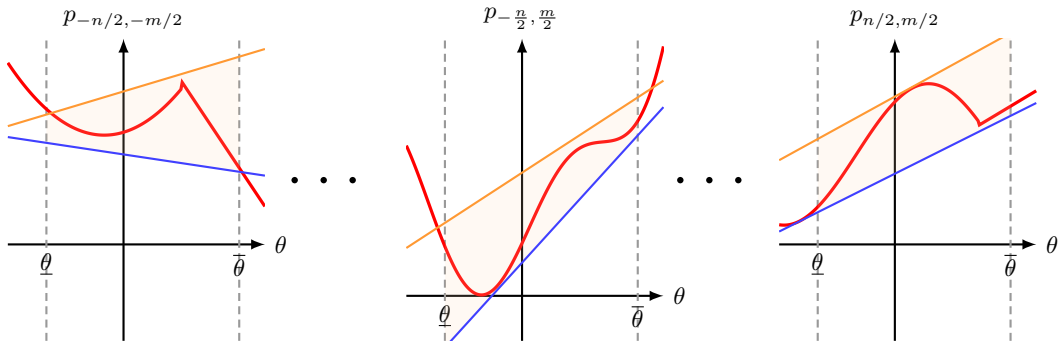


Figure 3.11: Linear relaxations of pixel values $p_{-n/2,-m/2}$, $\ldots$, $p_{-\frac{n}{2},\frac{m}{2}}$, $\ldots$ $p_{n/2,m/2}$ on $[\underline{\theta}, \overline{\theta}]$. Each function p_{ij} is enclosed between an upper affine bound $\overline{a} \cdot \theta + \overline{b}$ (orange) and a lower affine bound $\underline{a} \cdot \theta + \underline{b}$ (blue). The orange shaded band shows the approximation region of the relaxation.

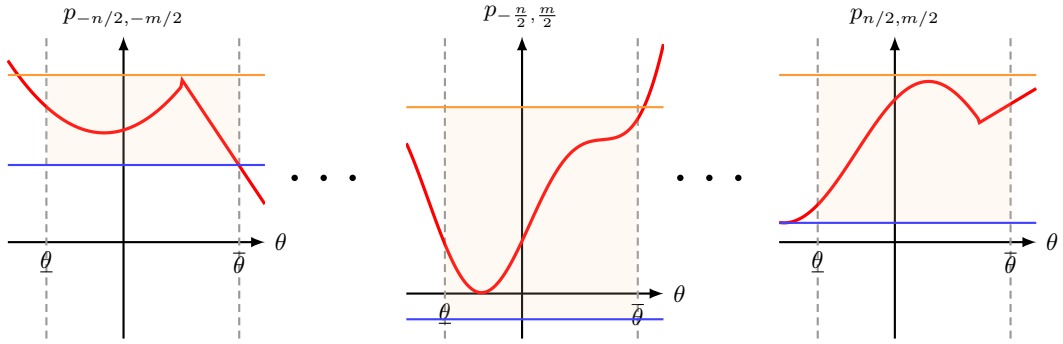


Figure 3.12: Constant relaxations of pixel values $p_{-n/2,-m/2}$, $\ldots$, $p_{-\frac{n}{2},\frac{m}{2}}$, $\ldots$ $p_{n/2,m/2}$ on $[\underline{\theta}, \overline{\theta}]$. Each function p_{ij} is enclosed between an upper constant bound $\overline{p_{i,j}}$ (orange) and a lower constant bound $\underline{p_{i,j}}$ (blue). The orange shaded band shows the approximation region of the relaxation.

Remark 26 (Perturbation as a new layer)**.** *As explained in the previous subsection 2.4.4 on brightness perturbation, to leverage the relaxation techniques, we reorganise the network to consider as input the transformation parameter κ and the picture x as a constant. Indeed, we deal with the geometric transformation g_x not as an external perturbation, but as the first layer of a composite neural network. The verification problem is thus performed on the composition $f \circ g_x$, as illustrated in Figure 3.13.*

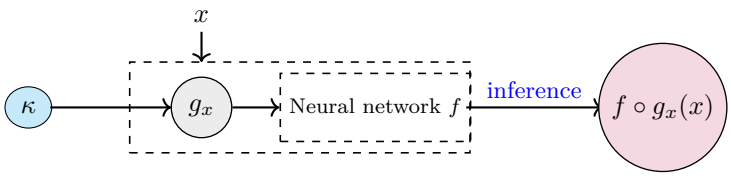
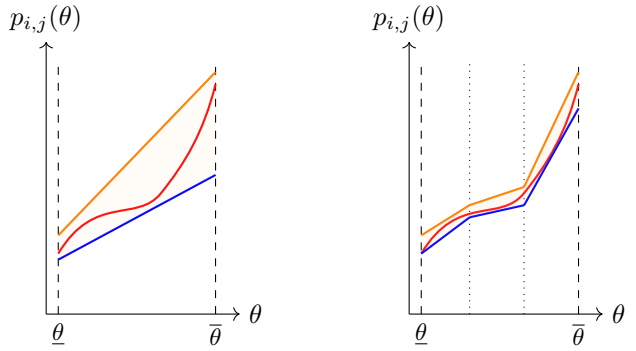


Figure 3.13: The verification problem is performed on the composition $f \circ g_x$ model.

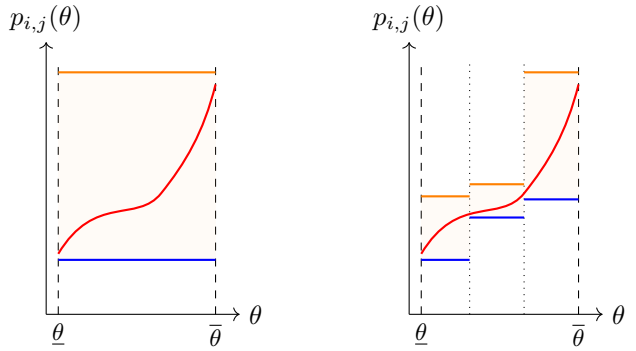
3. **Precision of the bounds.** The precision of the relaxation in Equation (3.5) strongly depends on the size of the range of perturbations $[\underline{\kappa}, \overline{\kappa}]$ on which it is computed. By reducing the range of perturbations, the local behavior of the geometric transformation is increasingly linear. As illustrated in Figure 3.14 and in Figure 3.15. for a rotation by an angle θ, this allows the upper and lower bounds to follow the curve much more closely, significantly reducing the over-approximation area.

   We introduce a hyperparameter, named the *interval size* (IS), that controls how



(a) Single large interval (b) Three sub-intervals

Figure 3.14: Impact of the interval size on relaxation.



(a) Single large interval (b) Three sub-intervals

Figure 3.15: Comparison of constant relaxations on single vs. split intervals.

the range of perturbations is split into sub-intervals. We divide the range of perturbation $[\underline{\theta}, \overline{\theta}]$ into N sub-intervals $[\theta_k, \theta_{k+1}]$ such that $\theta_{k+1} - \theta_k = \text{IS}$. The verification is then successful if and only if it holds for every sub-interval.

Example 11 (Interval size)**.** *Consider a range of perturbation* $[0°, 30°]$ *and an interval size of* $10°$*. Then the range of perturbation is partitioned into 3 sub-intervals:* $[0°, 10°]$, $[10°, 20°]$, $[20°, 30°]$.

Remark 27 (Generalization to multi-dimensional parameter)**.** *In the general case where* $\kappa \in \mathbb{R}^d$*, we define an interval size for each dimension rather than using the same interval size for all dimensions.*

4. **Computational efficiency.** However, this gain in precision comes with a direct computational cost. If we split the original range of perturbations into N sub-intervals and verify each sub-interval separately, we must run the solver N times on $f \circ g_x$. In practice, this means that the total verification time grows proportionally with the number of sub-intervals. For instance, in Figure 3.14 (a), the perturbation range is treated as a single sub-interval, so the verification is applied to $f \circ g_x$ only once, whereas in (b) it is applied three times, once for each sub-interval.

This section formalized the verification problem induced by geometric perturbations, characterized the set of transformed pictures, and justified why an over-approximation is necessary. We have also highlighted that the tightness of this relaxation depends not only on the geometric transformation but also on the chosen interval size used to partition the range of perturbations.

In the following section, we review how existing formal verification methods manage the interval size trade-off between precision and computational cost, which relaxation techniques they employ to over-approximate the set of transformed pictures, and what method is applied to verify the neural network f based on this perturbation.

3.3 Formal methods for geometric robustness

In this section, we review state-of-the-art methods for certifying the geometric robustness of neural networks using formal verification methods.

State-of-the-art methods include DeepG Balunovic et al. [2019], Semantify-NN Mohapatra et al. [2020], FGV Yang et al. [2022], and PWL Batten et al. [2024]. These methods accept four inputs: the reference picture x, the neural network f, the geometric parameter range $[\underline{\kappa}, \overline{\kappa}]$, and the Interval Size (IS), which serves as a precision hyperparameter. They determine whether f is robust for all transformed pictures $g_x^{[\underline{\kappa},\overline{\kappa}]}$, as illustrated in Figure 3.16. Typically, these methods verify the reachability property for the classification task, following the definition provided in Subsection 2.2.2.

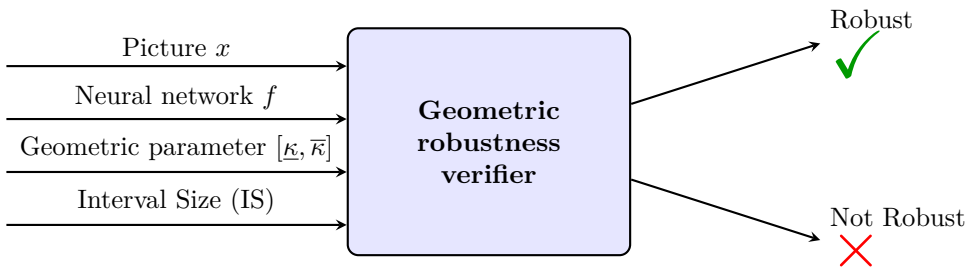


Figure 3.16: Overview of the verification process.

The geometric parameter range $[\underline{\kappa}, \overline{\kappa}]$ is partitioned into very small sub-intervals (defined by a the Interval Size IS):

$$[\underline{\kappa}, \overline{\kappa}] = \bigcup_l [\kappa_l, \kappa_{l+1}] \tag{3.9}$$

Each method evaluates each sub-interval $[\kappa_l, \kappa_{l+1}]$ independently, as depicted in Figure 3.17. For each sub-interval, an over-approximation of the set of transformed pictures, denoted as $g_x^{a[\kappa_l,\kappa_{l+1}]}$ (following Equation (3.6)) is computed. This over-approximation serves as the input domain for the relaxation of the neural network f; the specific relaxation (CROWN, IBP, or MILP) depends on the tool, as shown in the third column of Table 3.4. The final robustness guarantee is established if and only if the property holds for every sub-interval.

The choice of the relaxation requires a careful trade-off between precision and computational efficiency. Existing approaches generally fall into two distinct strategies. The first strategy treats g_x as a sequence of independent elementary operators (e.g., addition, division, sine, cosine, max, abs), bounds them individually and then composes them. The second strategy views g_x as a unique function and exploits its mathematical properties, such as Lipschitz continuity, to construct bounds directly. Table 3.4 categorizes state-of-the-art tools into either the operator-wise or Lipschitz-based strategy and summarizes the relaxation techniques used for both the geometric transformation g_x and the neural network f.

In the following, we detail these two main strategies.

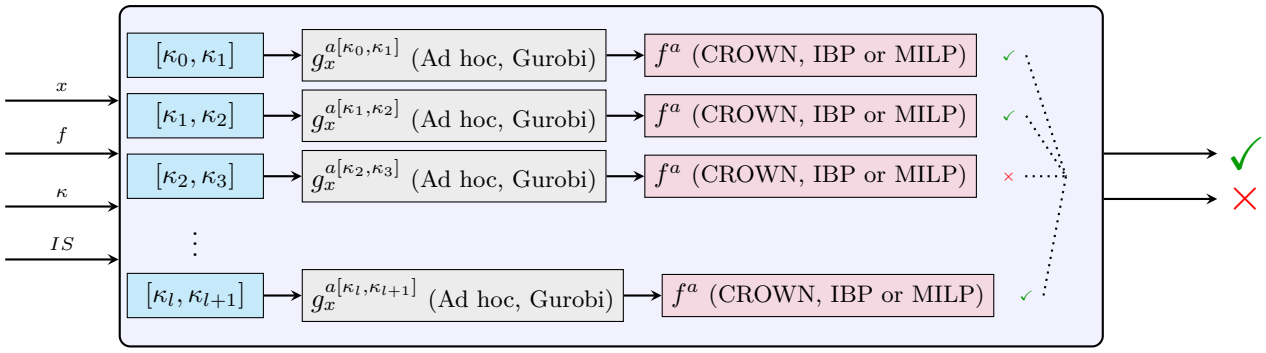


Figure 3.17: Verification methods workflow.

	g_x **bounding strategy**	g_x **relaxation (Bound type)**	g_x **tool**	f **relaxation (Bound type)**	f **tool**
⋆**S-NN**	Operator-wise	Constant	CROWN	Linear	Ad hoc (CROWN)
FGV	Operator-wise	Constant	Ad hoc	Constant	auto-LiRPA (IBP)
DeepG	Lipschitz	Linear	Gurobi	Linear	ERAN (CROWN)
PWL	Lipschitz	Piecewise linear	Gurobi	Convex polytope	VENUS (MILP)

Table 3.4: Comparison of state-of-the-art geometric verification tools. ⋆S-NN is Semantify-NN.

3.3.1 Strategy 1: Operator-wise Bounding (IBP-based)

The geometric transformation g_x is a function composed of trigonometric functions, absolute values, and minimum/maximum operations (as seen in Equation (3.4)). To handle this, Strategy 1 decomposes the transformation and relaxes each elementary operator individually using IBP (Subsection 2.3.5). Given that IBP is a loose method, the geometric parameter range $[\underline{\kappa}, \overline{\kappa}]$ is partitioned into very small sub-interval $[\kappa_l, \kappa_{l+1}]$ (Equation (3.9)). Methods following this strategy start by bounding the values of each pixel within each sub-interval as an interval following Equation (3.8):

$$\forall \kappa \in [\kappa_l, \kappa_{l+1}], \qquad \underline{p_{i,j}^l} \leq p_{i,j}(\kappa) \leq \overline{p_{i,j}^l} \tag{3.10}$$

The following subsections describe the specific calculation of $[\underline{p_{i,j}^l}, \overline{p_{i,j}^l}]$. Before looking at those details, we outline the steps (g_x relaxation and f relaxation) for both methods that employ this strategy, with a focus on their key differences:

- **Semantify-NN** Mohapatra et al. [2020] was one of the first methods developed for geometric robustness.
  - g_x **relaxation**: Semantify-NN computes $g_x^{a[\kappa_l,\kappa_{l+1}]}$, the over-approximation of transformed pictures on $[\kappa_l, \kappa_{l+1}]$ (Equation (3.6)) using IBP, illustrated in light gray in Figure 3.18. Then, Semantify-NN is the only approach that aggregates these local intervals by taking, for each pixel, the global minimum and maximum across all sub-intervals l:

    $$\left[\min_l \underline{p_{i,j}^l}, \ \max_l \overline{p_{i,j}^l}\right]$$

    The resulting over-approximation of the set of transformed pictures, denoted $g_x^{a[\underline{\kappa},\overline{\kappa}]}$ and illustrated in dark gray, serves as the input for the relaxation of the neural network f. g_x relaxation is achieved through an ad hoc implementation in Python.

– **Tool for f relaxation:** To relax f, Semantify-NN relies on an ad hoc Python implementation of CROWN rather than a solver (e.g `auto_LiRPA`). Finally, it provides a robustness guarantee for all transformed pictures $g_x(\kappa)$ over the full range $[\underline{\kappa}, \overline{\kappa}]$.

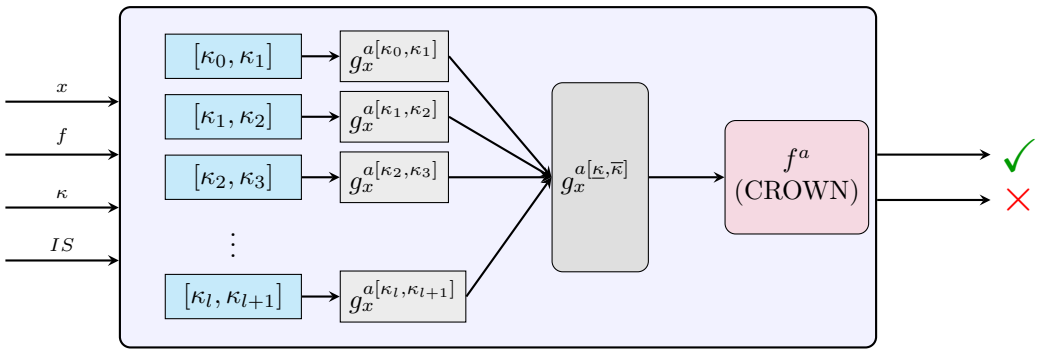


Figure 3.18: Semantify-NN verification workflow.

- **FGV** Yang et al. [2022] introduces a verification method prioritising speed.
  - g_x **relaxation**: FGV also computes $g_x^{a[\kappa_l,\kappa_{l+1}]}$. FGV maintains the independence of each sub-interval throughout the verification process, as illustrated in Figure 3.17. g_x relaxation is done with an ad hoc implementation in Python.
  - **Tool for f relaxation:** The relaxation method used for f is IBP, implemented via the `auto_LiRPA` solver. Since the forward pass of IBP is efficient, FGV parallelizes the verification of multiple pictures on the GPU.

Notation 5 (Simplified Notation)**.** *For the sake of clarity, in the following sections, the notation $[\underline{\kappa}, \overline{\kappa}]$ refers to the specific sub-interval currently under evaluation (formerly $[\kappa_l, \kappa_{l+1}]$). Similarly, $[\underline{p^l_{i,j}}, \overline{p^l_{i,j}}]$ is written $[\underline{p_{i,j}}, \overline{p_{i,j}}]$.*

We now present the technical details of FGV and Semantify-NN for obtaining the pixel relaxation $[\underline{p_{i,j}}, \overline{p_{i,j}}]$. There are two steps:

- Apply IBP to the geometric matrix $T_\kappa^{-1}(i,j)$.
- Apply IBP to the interpolation.

While the first step is common to both approaches, the second step differs. As discussed in the last subsection, their implementations also differ.

3.3.1.1 IBP to the geometric matrix

For a fixed coordinate (i,j) and a range of perturbations $[\underline{\kappa}, \overline{\kappa}]$, both methods begin by applying IBP to the transformation matrix $T_\kappa^{-1}(i,j)$. Since the parameter κ varies within range of perturbations, the result is not a single point (i',j') but a set of possible coordinates forming a trajectory. An over-approximation of this set is computed by bounding the elementary operators within the transformation matrix (such as trigonometric functions, addition, and multiplication). This yields a rectangle defined by:

$$([\underline{i'}, \overline{i'}], [\underline{j'}, \overline{j'}]) = T^{-1}_{[\underline{\kappa}, \overline{\kappa}]}(i,j) \tag{3.11}$$

Notation 6 (Coordinate domain)**.** *We define the coordinate domain by the rectangle $[\underline{i'}, \overline{i'}] \times [\underline{j'}, \overline{j'}]$, as illustrated in Figure 3.19.*

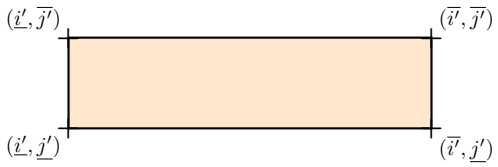


Figure 3.19: Coordinate domain defined by $[\underline{i'}, \overline{i'}] \times [\underline{j'}, \overline{j'}]$.

Figure 3.20 (a) illustrates the exact trajectory of the coordinate $(2, 0)$ over the range of perturbations $[0°, 70°]$ in red. Figure 3.20 (b) illustrates the IBP over-approximation of this trajectory with the coordinate domain, shown in orange.

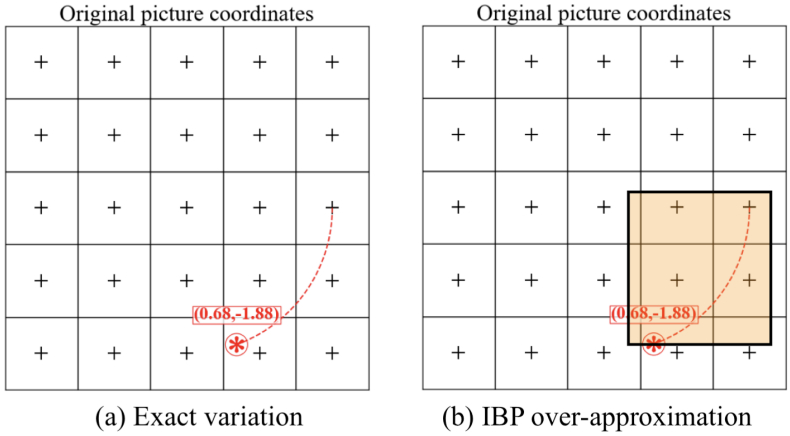


Figure 3.20: (a): The coordinate point $(i', j') = (0, 2)$ traces a red path over the range of perturbation $[\underline{\theta}, \overline{\theta}] = [0°, 70°]$. (b): IBP transforms it into a coordinate domain (orange rectangle).

Remark 28 (IS influences the coordinate domain)**.** *A smaller IS (Interval Size) results in a tighter coordinate domain.*

To apply IBP to the interpolation, the next steps are:

1. **Distance:** Evaluate the distance from each coordinate grid (i_k, j_k) to the four vertices of the coordinate domain $[\underline{i'}, \overline{i'}] \times [\underline{j'}, \overline{j'}]$.
2. **Relaxation:** Bound the pixel value over this area by an interval $[\underline{p_{i,j}}, \overline{p_{i,j}}]$.

We now examine how each method handles these steps.

3.3.1.2 IBP - Semantify-NN

We consider the coordinate domain computed by Semantify-NN and examine how $[\underline{p_{i,j}}, \overline{p_{i,j}}]$ is obtained.

- **Interpolation:** While most of the tools employ bilinear interpolation (Equation (3.1)), Semantify-NN implements *Inverse Distance Weighting* (IDW). The formula for a single coordinate value (i', j') with respect to all grid coordinates is:

$$I_x^{IDW}(i', j') = \frac{\sum_{k=1}^{n \times m} p_{i_k, j_k} \cdot \max(0, 1 - d_k(i', j'))}{\sum_{k=1}^{n \times m} \max(0, 1 - d_k(i', j'))} \tag{3.12}$$

$$\text{with } d_k(i', j') = \sqrt{(i' - i_k)^2 + (j' - j_k)^2} \quad \textit{Euclidean distance} \tag{3.13}$$

In Equation (3.12), the index k iterates over the entire image grid (from 1 to $n \times m$), where p_{i_k,j_k} is the value of the pixel at the k-th grid coordinate and d_k represents the Euclidean distance from that coordinate to the target point (i', j').

Remark 29 (IDW interpolation usage)**.** *While this technique is very common in Geographic Information Systems (GIS), it is unconventional in computer vision due to its higher computational cost compared to bilinear interpolation.*

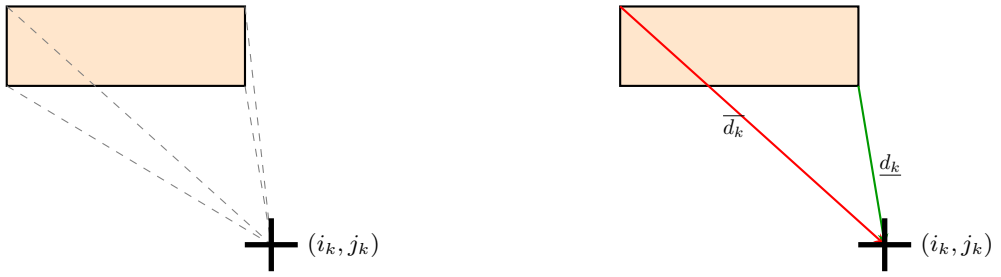
- **Relaxation method:** IBP is applied to the IDW formula (Equations (3.12) and (3.13)) over the coordinate domain.

  Remark 30 (Square root and IBP)**.** *The square roots of the Euclidian distance used by Semantify-NN in Equation* (3.13) *makes a direct IBP application difficult.*

  To bound the distance terms, the method evaluates the Euclidean distance (Equation (3.13)) from each coordinate grid (i_k, j_k) to the four vertices of the coordinate domain $[\underline{i'}, \overline{i'}] \times [\underline{j'}, \overline{j'}]$. The minimum and maximum of these values define the distance bounds $\underline{d_k}$ and $\overline{d_k}$ for each coordinate grid:

$$\underline{d_k} = \min(d_k(\underline{i}, \underline{j}),\ d_k(\underline{i}, \overline{j}),\ d_k(\overline{i}, \underline{j}),\ d_k(\overline{i}, \overline{j}))$$
$$\overline{d_k} = \max(d_k(\underline{i}, \underline{j}),\ d_k(\underline{i}, \overline{j}),\ d_k(\overline{i}, \underline{j}),\ d_k(\overline{i}, \overline{j}))$$

  Figure 3.21 illustrates, for a single grid coordinate (i_k, j_k), the four vertices distances (a) and the resulting distance bounds $(\underline{d_k}, \overline{d_k})$ (b).



(a) Evaluation of the four corner distances. (b) Resulting min/max distance bounds.

Figure 3.21: The method calculates distances to all four vertices of the coordinate domain (left) to the the coordinate grid (i_k, j_k) and determine the minimum and maximum bounds $\overline{d_k}$, $\underline{d_k}$ (right).

Finally, the pixel value bounds $[\underline{p_{i,j}}, \overline{p_{i,j}}]$ are obtained via IBP:

$$\overline{p_{i,j}} = \frac{\sum_{k=1}^{4} p_{i_k,j_k} \cdot \max(0, 1 - \underline{d_k})}{\sum_{k=1}^{4} \max(0, 1 - \overline{d_k})}, \qquad \underline{p_{i,j}} = \frac{\sum_{k=1}^{4} p_{i_k,j_k} \cdot \max(0, 1 - \overline{d_k})}{\sum_{k=1}^{4} \max(0, 1 - \underline{d_k})}$$

In practice, the computation of Semantify-NN is slow. This inefficiency stems from two main factors: the choice of the interpolation method (as noted in Remark 29) and the sequential computation of pixel values.

The Fast Geometric Verification (FGV) method addresses the scalability limitations of Semantify-NN by introducing a faster, fully parallelized implementation.

3.3.1.3 IBP - FGV

Given a coordinate domain $([\underline{i'}, \overline{i'}],\ [\underline{j'}, \overline{j'}])$, we examine how $[\underline{p_{i,j}}, \overline{p_{i,j}}]$ is obtained.

- **Interpolation:** FGV assumes standard bilinear interpolation that relies on linear distances along the axes:

$$I_x(i', j') = \sum_{k=1}^{n \times m} p_{i_k, j_k} \cdot \max\left(0, 1 - d_k^i(i')\right) \cdot \max\left(0, 1 - d_k^j(j')\right) \tag{3.14}$$

$$\text{with} \quad d_k^i(i') = |i' - i_k| \qquad d_k^j(j') = |j' - j_k| \tag{3.15}$$

Equation (3.14) applies independently to each channel. For ease of expression, we write p_{i_k, j_k} the pixel value at position (i_k, j_k) of an arbitrary channel.

- **Relaxation method:** IBP is applied to the bilinear interpolation formula (Equations (3.14) and (3.15)) over the coordinate domain. Equation (3.15) enables efficient computation of IBP bounds for the distance, since IBP for subtraction and the absolute value is straightforward. Indeed, the distance bounds are obtained by evaluating the absolute difference between the coordinate domain $([\underline{i'}, \overline{i'}], [\underline{j'}, \overline{j'}])$ and the integer grid coordinates (i_k, j_k). The distance bounds along the x-axis are defined as:

$$[\underline{d_k^i}, \overline{d_k^i}] = \begin{cases} [\underline{i'} - i_k, \ \overline{i'} - i_k] & \text{if } \underline{i'} - i_k \geq 0 \\ [i_k - \overline{i'}, \ i_k - \underline{i'}] & \text{if } \overline{i'} - i_k \leq 0 \\ [0, \max(|\underline{i'} - i_k|, |\overline{i'} - i_k|)] & \text{otherwise} \end{cases} \tag{3.16}$$

Analogous distance bounds, $\underline{d_k^j}$ and $\overline{d_k^j}$, are obtained for the y-axis.

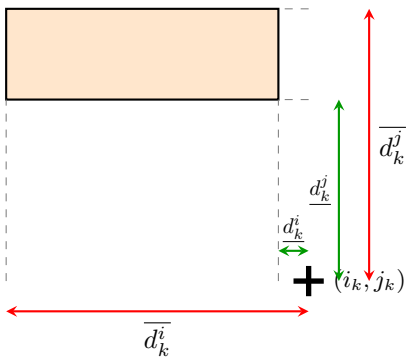


Figure 3.22: Distance bounds calculation for FGV. Distances are computed independently along the i and j axes to the edges of the coordinate domain, following Equation (3.16).

Remark 31. *FGV removes the complex square roots of the Euclidean distance used by Semantify-NN.*

These distances are then used to compute the weights:

$$\underline{w_k^i} = \max(0, 1 - \overline{d_k^i}) \quad \text{and} \quad \overline{w_k^i} = \max(0, 1 - \underline{d_k^i}) \tag{3.17}$$

The y-axis weights, $\underline{w_k^j}$ and $\overline{w_k^j}$, are determined in the same manner. Finally, the resulting pixel value interval $[\underline{p_{i,j}}, \overline{p_{i,j}}]$ is calculated as:

$$\underline{p_{i,j}} = \sum_{k=0}^{n \times m} p_{i_k, j_k} \cdot \underline{w_k^i} \cdot \underline{w_k^j} \qquad \overline{p_{i,j}} = \sum_{k=0}^{n \times m} p_{i_k, j_k} \cdot \overline{w_k^i} \cdot \overline{w_k^j} \tag{3.18}$$

While these equations define the theoretical bounds, implementing them efficiently requires addressing the large number of pixels.

3.3.1.4 Implementation details - Semantify-NN and FGV

A pixel only contributes to the interpolation if it is within a distance of 1 from the coordinate domain ($\max(0, 1 - d)$ term in interpolation Equations (3.12) and (3.14)). As shown in Figure 3.23, this means only 8 out of 25 pixels contribute to the exact transformation. For the IBP over-approximation it corresponds to 12 contributing pixels out of 25, so 48%. However, for an MNIST picture of 784 grid coordinates, this would represents only 0.76% (12/784) of contributing pixels.

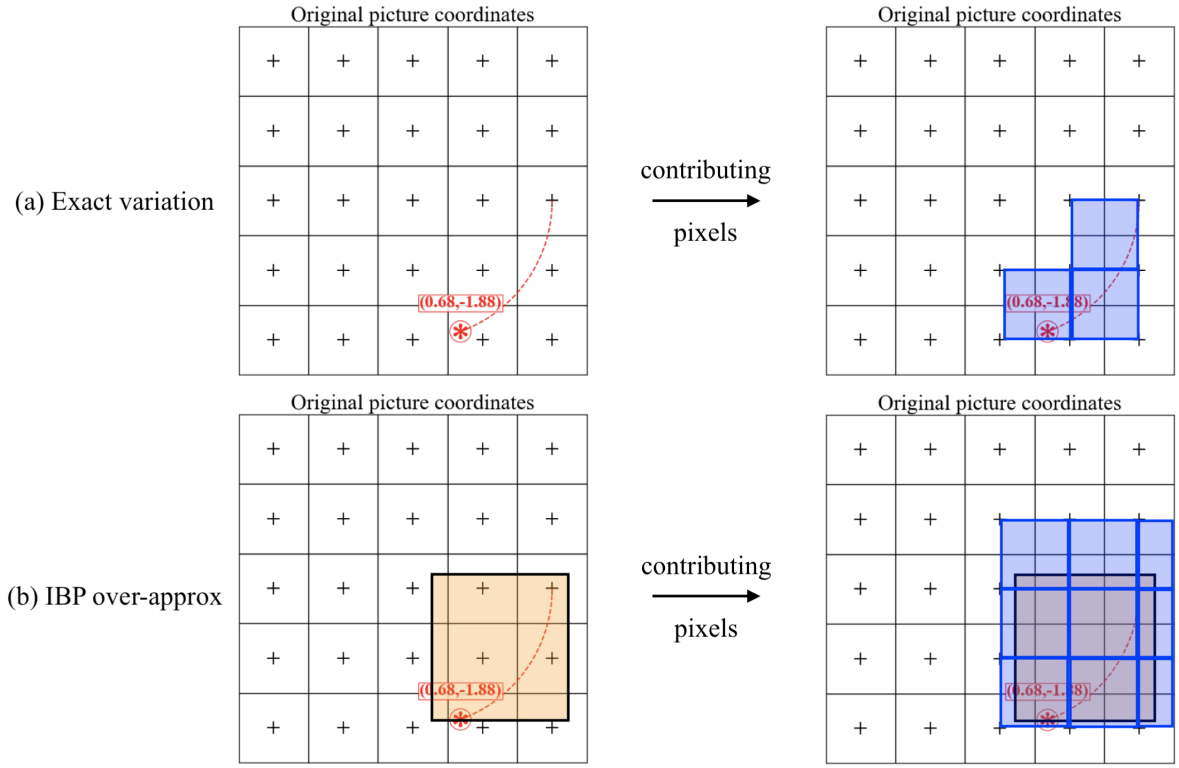


Figure 3.23: Contributing pixels

To handle these zero-weight pixels efficiently, Semantify-NN and FGV use the following optimizations:

- Instead of calculating IBP equations for every grid coordinate, Semantify-NN only selects pixels that are close enough to the coordinate domain to actually affect the result.
- FGV assigns a weight matrix to each transformed pixel matching the dimensions of the original input and leverages a fully vectorized approach where operations are performed on tensors, allowing the method to compute interval bounds for the entire picture grid in parallel on the GPU.

  Remark 32 (Sparse matrix operations)**.** *Storing a full weight matrix per coordinate would be memory-prohibitive. FGV addresses this by employing sparse matrix operations, a technique recording only non-zero values and their indices. A detailed and pedagogical explanation is provided in Appendix C.1 Yang et al. [2022].*

FGV implements two other optimizations to improve efficiency:

- Furthermore, the authors decompose bilinear interpolation into two distinct steps:

1. First, the interpolation weights (Equation (3.16) and Equation (3.17)) are obtained by computing the distances between the transformed coordinates and the grid coordinates along each axis. This process is independent of the pixel values. Therefore, doing this step for each picture is unnecessary. Indeed, FGV computes the interpolation weights **only once** for a given transformation, and they are valid for all pictures of the same size.
2. Second, the final pixel values (Equation (3.18)) are computed by applying these weights to the input picture via the bilinear formula.

- FGV employs batch parallelization processing multiple pictures simultaneously on the GPU.

FGV achieves remarkable speed via GPU optimization and simple IBP relaxations.

3.3.1.5 Comparison and Conclusion

While both methods employ IBP, FGV has significant advantages over Semantify-NN. Due to its scalability and effectiveness, it is widely considered the preferred baseline for comparison in this domain. Firstly, regarding **computational efficiency**, FGV is fully parallelized on the GPU across pictures and across pixel coordinates. Table 3.5 from Yang et al. [2022] illustrates this performance gap by comparing the runtime of interval rotation bound computations and network propagations of the sequential CPU-based Semantify-NN against the GPU-optimized FGV.

	Time to Compute Bounds (s)		
Dataset	Semantify-NN	FGV	FGV Speedup
MNIST	33.10	0.0029	11250×
CIFAR10	22.81	0.0026	8645×
Tiny ImageNet	62.83	0.0030	20610×

Table 3.5: Time to compute interval rotation bounds and network outputs for one batch of pictures. Semantify-NN computations are performed on the CPU, while FGV is performed on the GPU. The experiments were conducted on a machine with a 2.40GHz 24-core Intel Xeon Silver 4214R CPU and one Nvidia A100 GPU. Table taken from Yang et al. [2022].

Secondly, in terms of precision, both FGV and Semantify-NN achieve similar precision for the same IS. However, since FGV is faster, it can use a smaller interval size (IS) and thus achieve higher precision, as explained in Subsection 3.2.2.

Now that we have examined in detail the operator-wise IBP-based strategy, we turn to the second Lipschitz optimization strategy.

3.3.2 Strategy 2: Lipschitz optimization

The previous operator-wise strategy focused on *how* the picture is transformed step by step. It broke the transformation down into individual operators to compute the bounds. In contrast, this strategy treats g_x as a single function and focuses solely on the resulting pixel values to obtain an over-approximation of the transformed pictures.

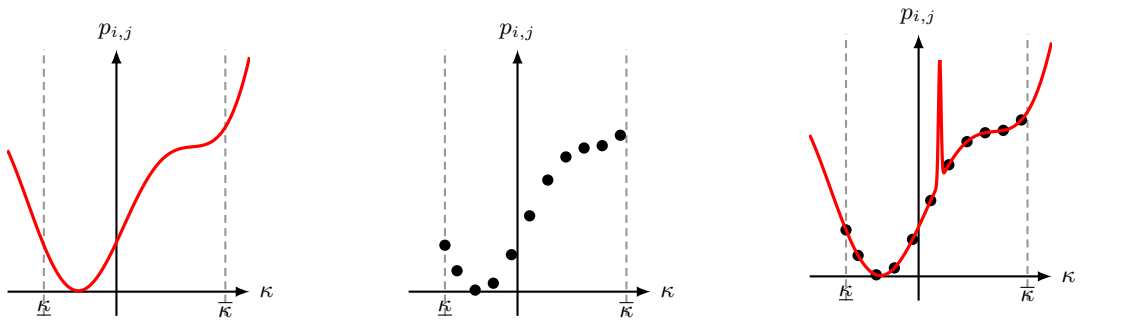
The perturbation domain $[\underline{\kappa}, \overline{\kappa}]$ is partitioned into sub-intervals (defined using the hyperparameter Interval Size): $[\underline{\kappa}, \overline{\kappa}] = \bigcup_l [\kappa_l, \kappa_{l+1}]$. Existing methods evaluate each sub-interval $[\kappa_l, \kappa_{l+1}]$ independently. They determine an over-approximation of $g_x^{[\kappa_l, \kappa_{l+1}]}$, which serves as the input for the relaxation of f. The robustness guarantee is given per sub-interval $[\kappa_l, \kappa_{l+1}]$.

Strategy 2 relies on two pillars: sampling and Lipschitz continuity. We first detail them. Next, we show how to apply them to construct an over-approximation of g_x over $[\underline{\kappa}, \overline{\kappa}]$. Finally, we compare the existing methods that implement this approach.

3.3.2.1 Sampling and Lipschitz continuity

The goal of this strategy is to bound the pixel value $p_{i,j}(\kappa)$ variation under $[\underline{\kappa}, \overline{\kappa}]$. As explained in Section 3.2.2, its exact trajectory is complex (Figure 3.24 (a)).

Fortunately, generating samples from this function is computationally cheap (Figure 3.24 (b)). However, as shown in Figure 3.24 (c), the function could exhibit extreme variations between two sampled points. There will never be a fine enough sampling to ensure that the function does not vary significantly between two samples.



(a) Exact variation

(b) Sampling

(c) Arbitrary variation consistent with samples

Figure 3.24: The functions in both (a) and (c) pass through the same sample points (b).

Indeed, bounding a function based solely on finite samples is not *sound*, meaning that there is no guarantee that the function does not exceed the estimated bounds between the sampled points, as defined in Definition 3.2.

Definition 3.2 (Sound bound)**.** *A bound is considered sound if it is guaranteed to enclose all possible values of a function over a given interval: the function will never exceed an upper bound, or drop below a lower bound.*

However, geometric transformations are *Lipschitz continuous* and their Lipschitz constants are analytically tractable to compute. This provides a mathematical property to bound the behavior of the function between sampled points.

The Lipschitz constant, denoted $\mathcal{L}_{i,j}$ (specific to each pixel), quantifies the maximum rate at which the function can change Heinonen [2001]. It is any upper bound on the supremum of the norm of the gradient of $p_{i,j}$ over the domain $[\underline{\kappa}, \overline{\kappa}]$. This yields:

$$\forall \kappa_1, \kappa_2 \in [\underline{\kappa}, \overline{\kappa}], \quad \|p_{i,j}(\kappa_2) - p_{i,j}(\kappa_1)\| \leq \mathcal{L}_{i,j} \|\kappa_2 - \kappa_1\| \tag{3.19}$$

Remark 33 (Lipschitz constant per dimension)**.** *Equation* (3.19) *assumes that $\mathcal{L}_{i,j}$ is a global Lipschitz constant over the domain $[\underline{\kappa}, \overline{\kappa}]$, meaning it bounds the norm of the transformation's gradient for each dimension. Note that similar properties apply if we choose a distinct Lipschitz constant for each dimension.*

To simplify the visualization, we restrict our analysis to a 1D parameter space, where the norm reduces to the absolute value $|\cdot|$. Consider a specific reference point $(\kappa_1, p_{i,j}(\kappa_1))$. We can rewrite Equation (3.19), we obtain upper and lower constraints for any κ:

$$p_{i,j}(\kappa) \leq p_{i,j}(\kappa_1) + \mathcal{L}_{i,j}|\kappa - \kappa_1| \tag{3.20}$$

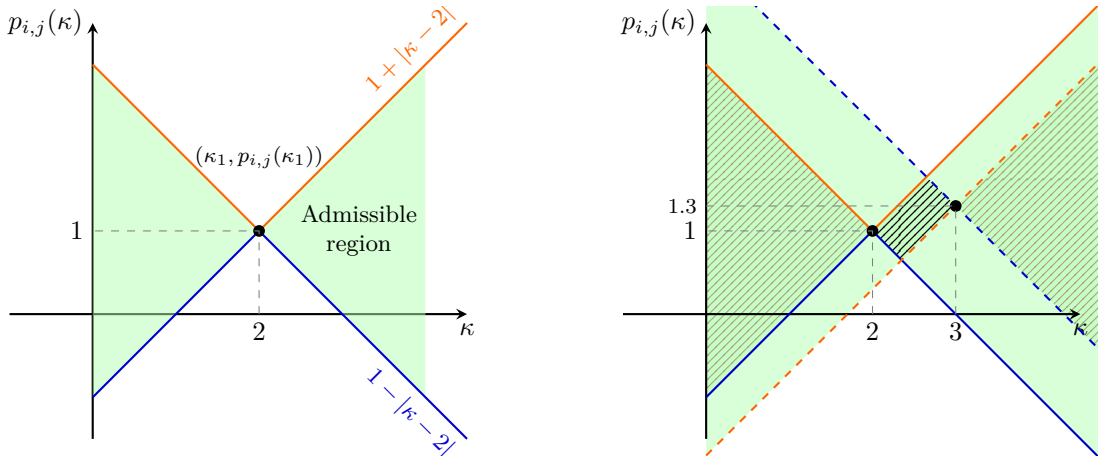
$$p_{i,j}(\kappa) \geq p_{i,j}(\kappa_1) - \mathcal{L}_{i,j}|\kappa - \kappa_1| \tag{3.21}$$

Definition 3.3 (Admissible region)**.** *The admissible region is the set of all points that satisfy the Lipschitz constraints imposed by Equations* (3.20) *and* (3.21)*. It represents the area enclosed between the upper and lower constraints. The variation of $p_{i,j}$ must lie entirely within this admissible region.*

Property 3.1 (Intersection of constraints)**.** *When multiple sample points are available, the admissible region of the function corresponds to the intersection of admissible regions defined by the Lipschitz constraints for each of these points.*

Example 12 (Lipschitz constraints)**.** *Consider a specific pixel where, for $\kappa_1 = 2$ we have a value $p_{i,j}(2) = 1$, and the Lipschitz constant under $[\underline{\kappa}, \overline{\kappa}]$ is $\mathcal{L}_{i,j} = 1$. We plot the upper constraint $\kappa \mapsto 1 + |\kappa - 2|$ from Equation* (3.20) *in orange and the lower constraint $\kappa \mapsto 1 - |\kappa - 2|$ from Equation* (3.21) *in blue.*

As illustrated in Figure 3.25 (a), the admissible region (shaded green) is the region between these two constraints. As shown in Figure 3.25 (b), introducing a second sample point tightens the admissible region (hatched area), following Property 3.1.



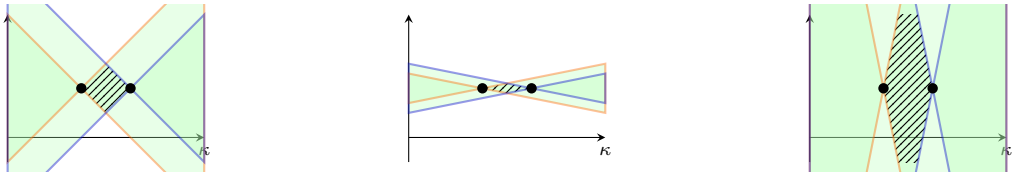
(a) Single reference point constraints

(b) Intersection of constraints from two points

Figure 3.25: Visualizing lipschitz constraints. (a) Admissible region from a single point (shaded green). (b) Admissible region from two points (darker green and hatched area).

We are now focusing on the admissible region **between** the two points. The intersection is now displayed in dark green, the region between the two points is hatched. Two key factors determine the size of this region:

1. **The Lipschitz constant ($\mathcal{L}_{i,j}$):** This constant dictates the slope of the constraints. A smaller $\mathcal{L}_{i,j}$ results in flatter slopes and a tighter admissible region, as shown in Figure 3.26.



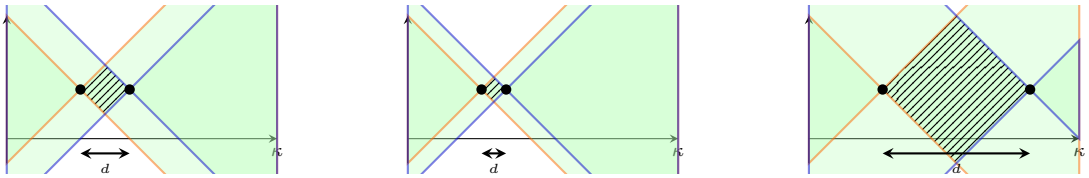
(a) $\mathcal{L}_{i,j} = 1$

(b) $\mathcal{L}_{i,j} = 0.2$

(c) $\mathcal{L}_{i,j} = 5$

Figure 3.26: Impact of the Lipschitz constant $\mathcal{L}_{i,j}$. A smaller $\mathcal{L}_{i,j}$ (b) restricts the admissible region compared to a larger $\mathcal{L}_{i,j}$ (c).

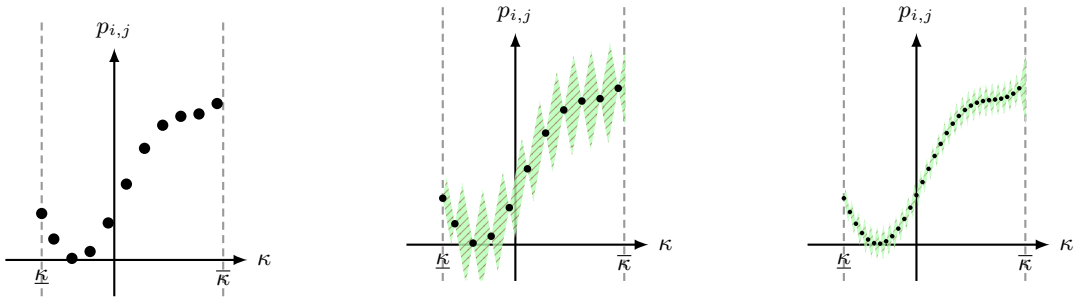
2. **The sampling density:** This is the distance $|\kappa_1 - \kappa_2|$ between sample points. Bringing sample points closer together reduces the admissible region, thereby increasing precision, as shown in Figure 3.27.



(a) $|\kappa_1 - \kappa_2| = 1$ (b) $|\kappa_1 - \kappa_2| = 0.5$ (c) $|\kappa_1 - \kappa_2| = 3$

Figure 3.27: Impact of the sampling distance. A small distance (b) minimizes the admissible region (hatched), compared to a large distance (c). $\mathcal{L}$ is fixed for (a) (b) (c).

As illustrated in Figure 3.28, the extreme variations hypothesized in Figure 3.24 (c) are impossible with a Lipschitz constant less than 5.5, as the function is strictly confined to the admissible region. This figure also illustrates the impact of the sampling density: adding samples leads to tighter admissible regions, as visible when comparing (b) to (c).



(a) 11 samples (b) Admissible region with 11 samples (c) Admissible region with 30 samples

Figure 3.28: Impact of the sampling density on the admissible region, with $\mathcal{L} = 5.5$.

We have established two observations. First, generating sample points is easy. Second, the Lipschitz continuity bounds the function variation in an *admissible region*. Adding more points makes this region smaller. Strategy 2, *Lipschitz optimization*, builds on these observations. We now move to the practical application.

3.3.3 Strategy steps

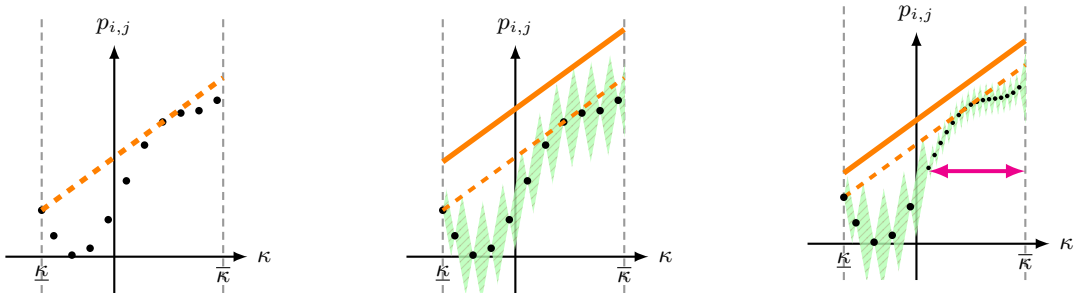
The strategy consists of three steps:

1. **Initial unsound bound:** The first step constructs a candidate bound calculated based on a finite set of sampled points $p_{i,j}(\kappa_1), \ldots, p_{i,j}(\kappa_P)$. As shown in Figure 3.29 (a), an initial linear function (in orange dashed) is an upper bound of the samples. However, this bound is **unsound**. Between the samples, there is no guarantee that the exact variation may exceed this bound.
2. **Lipschitz constant computation:** The second step is to compute a Lipschitz constant under the range of perturbations $[\underline{\kappa}, \overline{\kappa}]$.
3. **Correction for soundness:** To guarantee soundness, this bound requires correction. The *admissible region*, defined by the Lipschitz constant, determines the required correction term.
   For an upper bound, the unsound bound shifts upwards until it lies strictly above the admissible region. This ensures the bound remains valid for all κ, not just the sampled points. Figure 3.29 (b) illustrates this corrected, sound bound (orange)

above the admissible region, the green hatched area. Similarly, the lower bound is shifted downwards to lie strictly below the admissible region.

Adaptive sampling: Adding samples tightens the admissible region, as shown previously in Figure 3.27. However, samples are not necessary everywhere. They are required only in "regions of interest", areas where the admissible region forces the bound upward.

For instance in Figure 3.29 (c), adding samples in the area marked by the pink arrow reduces the height of the admissible region. Conversely, adding samples to the left of the arrow provides no benefit, as the admissible region there already lies well below the bound.



(a) Step 1: Unsound bound

(b) Step 3: Corrected sound bound

(c) Adaptative sampling for sound bound

Figure 3.29: Visualizing the three-step strategy, the second step is not display. (b) A linear unsound bound (a) is shifted upward until it clears the admissible region (hatched area). (c) Adding samples in the region of interest (pink arrow) reduces the admissible region's height, allowing for a tighter sound bound.

Existing methods using this strategy rely on three inputs to over-approximate $p_{i,j}^{[\underline{\kappa},\overline{\kappa}]}$:

- A set of P uniform samples $p_{i,j}(\kappa_1), \ldots, p_{i,j}(\kappa_P)$ to construct the initial unsound bound.
- A function to compute the Lipschitz constant $\mathcal{L}_{i,j}$, which defines the admissible region and ensures the bound is sound.
- A sampling function $p_{i,j}(\cdot)$ to execute adaptive sampling.

To bridge the gap between the high-level verification and the specific pixel-level relaxation, Figure 3.30 illustrates the simplified workflow for a single sub-interval. The pixel-level relaxation is depicted in Figure 3.31.

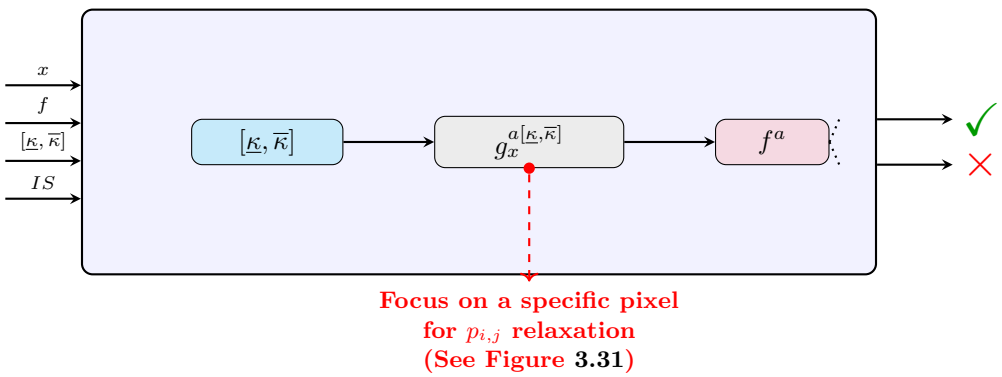


Figure 3.30: Verification workflow for a single sub-interval $[\underline{\kappa}, \overline{\kappa}]$. The geometric relaxation g_x^a is composed of individual pixel relaxations $p_{i,j}$.

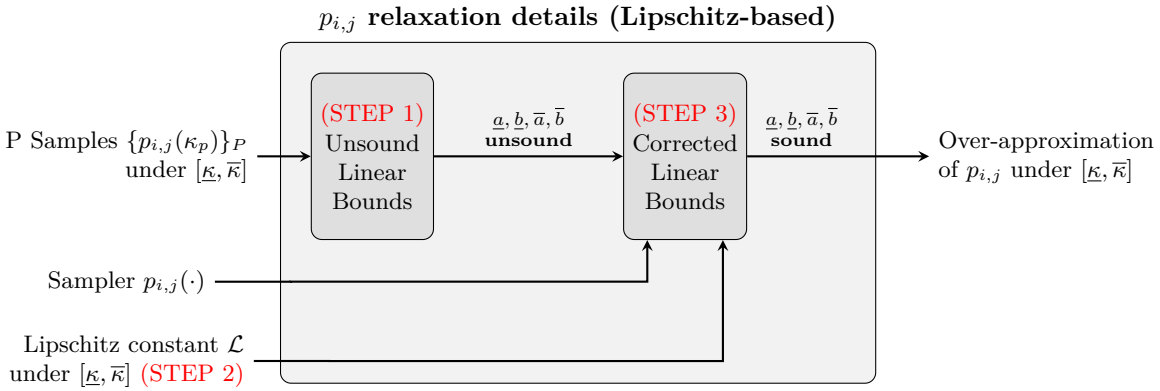


Figure 3.31: Detailed workflow of the Lipschitz-based relaxation.

3.3.3.1 Existing methods

Two methods utilize this strategy:

- **DeepG** Balunovic et al. [2019] was one of the first methods to address geometric robustness (released concurrently with Semantify-NN).
  - g_x **relaxation:** DeepG computes a linear relaxation of $g_x^{[\underline{\kappa}, \overline{\kappa}]}$, as shown in Figure 3.32 (a).
  - **Tool for f relaxation:** To relax the network f, DeepG uses DeepPoly (similar to CROWN) via the ERAN Müller et al. [2021] solver.
- **PWL** (Piecewise Linear) Batten et al. [2024] builds upon DeepG with the goal of improving precision.
  - g_x **relaxation:** Instead of a single linear bound, PWL constructs a 2-piecewise linear over-approximation. As shown in Figure 3.32 (b), this shape follows the non-convex variations of the pixel value much more closely than a simple linear bound.
  - **Tool for f relaxation:** Because the input domain is defined by a complex piecewise linear shape, PWL cannot use linear solvers like ERAN, DECOMON Ducoffe [2022] or auto-LiRPA. Instead, it utilizes VENUS, a Mixed-Integer Linear Programming (MILP) solver.



(a) Linear Relaxation

(b) Piecewise Linear Relaxation

Figure 3.32: Comparison of Linear Relaxation (left) vs. Piecewise Linear Relaxation (right). The orange shaded area represents the over-approximation of the function behavior.

The technical details of DeepG and PWL are deferred to the following section dedicated to our contribution, SuperDeepG. Since our method builds directly upon

DeepG, we will describe the method step-by-step to explicitly highlight our improvements at each stage.

3.3.3.2 Comparison

The Lipschitz optimization strategy proposed by DeepG offers a theoretically sound way to handle global geometric transformations. PWL significantly improves upon DeepG's precision by adopting a piecewise linear domain, as illustrated in Figure 3.32.

However, this precision comes at a cost. The complex piecewise linear input domain requires the use of a MILP solver (VENUS), which is computationally expensive and does not scale well to large networks. Regarding computation, DeepG relies on a 2019 CPU implementation, which is relatively slow compared to modern GPU-accelerated verifiers.

Having detailed both the Operator-wise Bounding and Lipschitz optimization strategies, we now proceed to a comparative analysis. Understanding the trade-offs between these approaches, specifically regarding precision, computational efficiency, is essential for identifying the limitations of current state-of-the-art tools and motivating the design of our proposed method, Super-DeepG.

3.4 Comparison of Operator-wise and Lipschitz-based Strategies

While both strategies share the goal of certifying geometric robustness, they approach the relaxation of the geometric transformation g_x very differently. In this section, we compare the Operator-wise Bounding strategy (detailed in Subsection 3.3.1, denoted Strategy 1) and the Lipschitz optimization strategy (detailed in Subsection 3.3.2, denoted Strategy 2). The key differences are summarized below (Table 3.6):

- **Treatment of non-linearity (Decomposition vs. global view):** Strategy 1 (Operator-wise) addresses non-linearity by decomposing the geometric transformation g_x into a sequence of elementary operators. While this simplifies the implementation, over-approximation errors accumulate at every intermediate step (e.g., sine, cosine, multiplication). In contrast, Strategy 2 (Lipschitz-based) treats g_x as a global function. By leveraging the Lipschitz constant $\mathcal{L}_{i,j}$, it bounds the output directly based on input variations. This mitigates intermediate error accumulation. The following remark 34 further highlights an additional advantage of Strategy 2.

  Remark 34 (Independence from interpolation complexity)**.** *Strategy 2 places no constraints on the complexity of the interpolation or the operators used, as long as they are differentiable to compute the Lipschitz constant. This avoids the limitations of the operator-wise strategy, such as the difficulty of handling Euclidean distance (as discussed in Subsection 3.3.1.3, Remark 30).*

- **Computational efficiency:** There is a gap in implementation between the two strategies. Strategy 1, exemplified by FGV, leverages modern GPU parallelization and vectorized tensor operations to process large batches of pictures efficiently. Conversely, existing implementations of Strategy 2 are either outdated (DeepG relies on CPU execution) or computationally prohibitive (PWL requires MILP solvers). Consequently, tools based on Strategy 1 currently offers significantly better efficiency.

- **Resulting input domain and network scalability:** The shape of the relaxed input domain $g_x^{[\underline{\kappa},\overline{\kappa}]}$ dictates which verification tools can be applied to the neural network f.
  - Strategy 1 typically produces *Hyper-rectangles*. This simple convex shape allows for fast propagation through large networks using standard IBP.
  - Strategy 2, particularly PWL, generates complex *Piecewise Linear* domains. While precise, handling such domains requires expensive solvers (like VENUS) that do not scale to large architectures.
  - Strategy 2, particularly DeepG, uses linear domains and CROWN for the network relaxation. It does not support CROWN-IBP (see Remark 16) for improved scalability.
- **Dependency on Training Methodology:** The performance of FGV is heavily coupled with its Certified Geometric Training (CGT) to align the network weights with IBP verification; without it, the verification is often too imprecise on standard networks. Strategy 2 is agnostic from the training process, aiming to verify standard networks.

Strategy	**Pros**	**Cons**
Strategy 1 (Operator-wise)	• Fast computation using GPU parallelization • Produces domains (hyper-rectangles) that are fast to verify and scales well to large neural networks	• Accumulates over-approximation at each step • Often requires specific training for the neural network • Struggles with complex distance formulas
Strategy 2 (Lipschitz-based)	• Treats the transformation as one global function: prevents the accumulation of intermediate over-approximation. • Works directly on standard neural networks (no training needed) • Handles any differentiable interpolation (if the Lipschitz constant can be computed)	• Requires heavy and slow computations • Does not scale to large neural networks

Table 3.6: Pros and cons of the two bounding strategies, based on the analysis of existing methods (Semantify-NN, FGV, DeepG, and PWL).

Conclusion and Motivation. Current methods present a dilemma: one must choose between the scalability of Strategy 1 (at the cost of decomposition errors) or the mathematical soundness of Strategy 2 (at the cost of computational inefficiency or loose linear bounds).

This analysis highlights an opportunity for a hybrid approach. An ideal method would leverage the Lipschitz optimization to avoid decomposition errors, while adopting the high-performance GPU engineering using linear domain shapes (a trade-off between IBP and piecewise linear) to ensure scalability. This observation forms the basis for our contribution, Super-DeepG, described in the following section.

Chapter 4

Overview of Super-DeepG approach

In this section, in order to introduce our proposed approach, Super-DeepG, we clarify and reformulate existing methods such as DeepG and PWL. As emphasized in Subsection 2.1.4 (Page 30) and Subsection 3.4 (Page 80), the central challenge in verifying robustness against geometric perturbations is the trade-off between precision and scalability/speed. Our comparison of the two existing strategies revealed a clear gap: operator-wise bounding is fast but imprecise, while Lipschitz optimization is precise but computationally expensive.

Our motivation is to demonstrate that Lipschitz-based method of DeepG should not be given up due to its computational cost. Instead, it requires a reformulation. As shown in Table 4.1, by adapting these mathematical foundations to couple with modern parallel hardware, Super-DeepG demonstrates that this strategy can be both precise and efficient. Consequently, it offers another direction for future research compared to methods that rely on imprecise approximations with specific training constraints, such as FGV.

$g_x(\cdot)$	**Operator-wise bounding strategy**		**Lipschitz optimization strategy**		
	semantify-NN	FGV	DeepG	PWL	**Super-DeepG**
Precise	✓	✓	✓✓	✓✓✓	✓✓
Fast/Scalable	✓	✓✓✓	✗	✗	✓✓

Table 4.1: Tools implementing geometric abstract domains

Our main contributions are as follows:

1. **Reformulation and clarification of DeepG:** DeepG (2019) was the pioneering method for the Lipschitz optimization strategy. However, the article focused on theory rather than implementation details. Consequently, the mathematical formulation was often implicit or only found within the source code, and the paper lacked illustrative figures. In this work, we provide a comprehensive reformulation of DeepG's methodology. We clarify the problem statement for each step, explicitly write out the resolution methods (including equations previously hidden in the code), and provide visualizations to facilitate understanding.

2. **Analytical resolution and speedup:** We develop Super-DeepG. We eliminate the use of iterative methods and solver of DeepG. Instead, we propose new analytical formulations for each optimization problem that achieve equivalent precision. This change removes a computational bottleneck and enables efficient GPU parallelization. Our experiments show a speedup of up to 526×. Furthermore, we achieve significant geometric certification results on TinyImageNet without training specifically for geometric robustness as opposed to FGV.

3. **Open-source implementation:** We provide an open-source implementation in `auto_LiRPA`. We encapsulate the geometric transformations in a `CustomOp` (defined in Subsection 2.4.3), which simplifies integration and usability for practitioners.

Like prior tools, Super-DeepG takes as input: the neural network f, the reference picture x, the perturbation range $[\underline{\kappa}, \overline{\kappa}]$, and the Interval Size (IS). It returns the verification result: Robust/Not Robust. As shown in Table 4.2, Super-DeepG uses linear relaxation for the geometric transformation g_x based on the Lipschitz optimization strategy (Subsection 3.3.2). The relaxation of $f \circ g_x$ is done using the `auto-LiRPA` verifier with the CROWN or CROWN-IBP.

g_x **bounding strategy**	g_x **relaxation (Bound type)**	g_x **tool**	f **relaxation (Bound type)**	f **tool**
Lipschitz optimization	Linear	`auto-LiRPA`	Linear	`auto-LiRPA`

Table 4.2: Super-DeepG geometric verifier (see Table 3.4 for the comparison of state-of-art approaches).

We structure this section according to the three steps of the Lipschitz optimization strategy identified in Subsection 3.3.2:

- **STEP 1:** Find the best fitting linear (unsound).
- **STEP 2:** Compute the Lipschitz constant.
- **STEP 3:** Shift the line to make it sound.

For each step i, we proceed as follows:

- **STEP** i**.A:** Introduce the problem statement and the notation used to resolve it.
- **STEP** i**.B:** Detail the resolution methods employed by DeepG and PWL.
- **STEP** i**.C:** Present Super-DeepG resolution using mathematical equations, algorithms, and code:
  - **1 dimension:** For any $\kappa \in \mathbb{R}$, corresponding to a one-dimensional transformation such as rotation, scaling, and shearing.
  - **multi-dimensions:** For $\kappa \in \mathbb{R}^d$ (e.g., $d = 2$ for a translation).

Finally, we conduct experiments comparing Super-DeepG, DeepG, PWL and FGV.

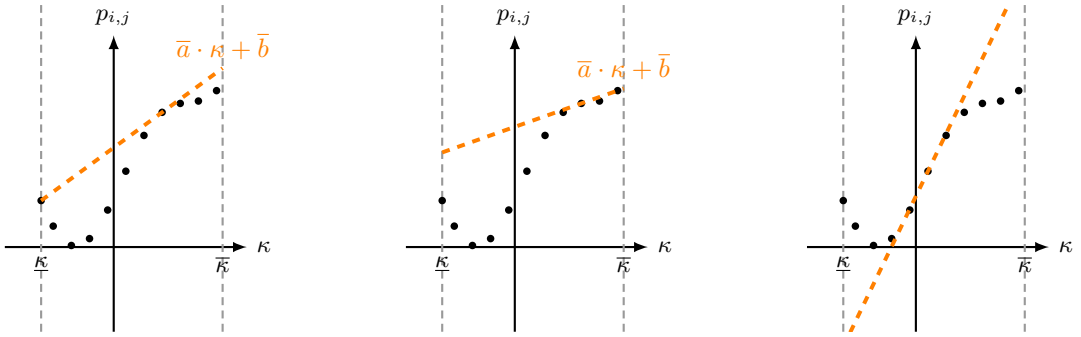
4.1 STEP 1: Unsound linear bounds

The objective of this step is to construct candidate linear bounds per pixel defined by coefficients ($\underline{a}, \overline{a} \in \mathbb{R}^d$ and $\underline{b}, \overline{b} \in \mathbb{R}$) based on a finite set of sample points $\{p_{i,j}(\kappa_p)\}_P$. Thus, these bounds are **unsound** because they are only guaranteed to hold for the sampled points. They will be corrected in Step 3 to ensure validity across the entire continuous domain.

4.1.1 STEP 1.A: Problem statement

We first present a geometric intuition of this step, illustrated in Figure 4.1, and then give the formal expression. We focus on the upper bound; a similar process is applied to compute the lower bound. To select the best bound, we evaluate candidates based on two criteria, **validity** and **tightness**:

- **Validity:** The orange dashed linear function must act as a valid upper bound over the set of samples. Geometrically, this means the line must lie **above** every sampled point. In Figure 4.1, candidates (a) and (b) satisfy this condition. In contrast, candidate (c) is rejected because it intersects the sample set, leaving some points above the line.



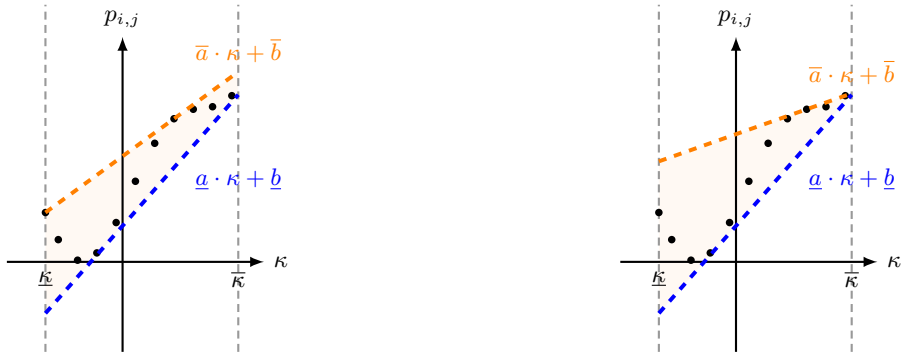
(a) Valid candidate (b) Valid candidate (c) Invalid candidate

Figure 4.1: Comparison of different linear candidates for the upper bound. Candidates (a) and (b) are valid as they lie above all samples. Candidate (c) is invalid as some sample points lie above the line.

- **Tightness:** Among the valid candidates, we seek the one that ensures the highest precision. We want to minimize the empty space between the linear bound and the samples. In Figure 4.1, while both (a) and (b) are valid, (a) seems to be preferable because it follows the shape of the sample more closely than (b). To evaluate this mathematically, as established in Definition 2.8 of Section 2.3.3, the tightness of a linear relaxation is evaluated by the area of its over-approximation:

$$Area = \int_{\underline{\kappa}}^{\overline{\kappa}} \Big((\overline{a} \cdot \kappa + \overline{b}) - (\underline{a} \cdot \kappa + \underline{b}) \Big) \, d\kappa \tag{4.1}$$

  Figure 4.2 visualizes this metric. Assuming a fixed lower bound, the over-approximation area (shaded in orange) is smaller for candidate (a) than for candidate (b). Therefore, the upper bound proposed in (a) is a tighter linear relaxation.



(a) Tighter bound (Candidate (a)) Area = 2.89 (b) Looser bound (Candidate (b)) Area = 3.58

Figure 4.2: Visualizing the tightness of valid candidates by the area of their over-approximation (shaded in orange).

As DeepG and PWL, we formalize these requirements using the concept of the **residual**. For a given linear upper bound defined by slope $\overline{a}$ and bias $\overline{b}$, the residual at a sample point κ is the signed vertical distance between the bound and the sample value:

$$\overline{r}(\kappa; \overline{a}, \overline{b}) = \Big(\overline{a} \cdot \kappa + \overline{b} \Big) - p_{i,j}(\kappa) \tag{4.2}$$

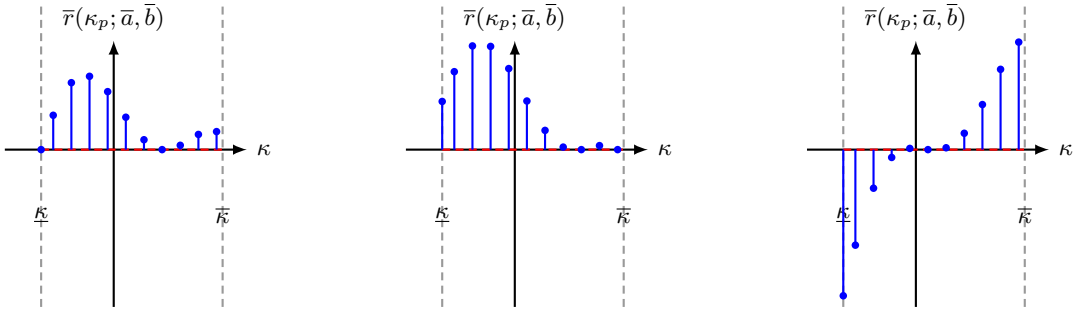
For the lower residual we have $\underline{r}(\kappa; \underline{a}, \underline{b}) = p_{i,j}(\kappa) - (\underline{a} \cdot \kappa + \underline{b})$. Using the residuals visualized in Figure 4.3, we can evaluate our criteria:

- **Validity:** A candidate is valid only if the residual is positive $\forall p \in \{1, \ldots, P\}$.

  - In Figures 4.3 (a) and (b), all residuals are positive, confirming validity.
  - In Figure 4.3 (c), we observe residuals taking negative values, confirming that this candidate violates the constraints.

- **Tightness:** Among valid candidates, we seek to minimize the **average residual** $\left(\frac{1}{P}\sum_{p=1}^{P}\overline{r}(\kappa_p;\overline{a},\overline{b})\right)$, as defined by Property 4.1.
  - For candidate (a) in Figure 4.3, the average residual is 0.12.
  - For candidate (b) in Figure 4.3, the average residual is 0.45.

  Since $0.12 < 0.45$, candidate (a) is preferred over candidate (b).



(a) All $\overline{r}(\kappa_p;\overline{a},\overline{b}) \geq 0$ (b) All $\overline{r}(\kappa_p;\overline{a},\overline{b}) \geq 0$ (c) Some $\overline{r}(\kappa_p;\overline{a},\overline{b}) \leq 0$

Figure 4.3: Residuals corresponding to the candidates in Figure 4.1.

Property 4.1 explains why we define this tightness requirement in terms of the average residual, as it directly approximates the over-approximation area given by Equation (4.1).

Property 4.1 (Average residual and over-approximation area)**.** *As established in Equation* (4.1)*, the tightness of a linear relaxation is evaluated by the area of its over-approximation. By adding and subtracting the pixel function $p_{i,j}(\kappa)$ inside the integral, we can split the total area into the sum of the upper residual $\overline{r}(\kappa;\underline{a},\underline{b})$ and the lower residual $\underline{r}(\kappa;\underline{a},\underline{b})$:*

$$\begin{aligned} Area &= \int_{\underline{\kappa}}^{\overline{\kappa}} \Big((\overline{a}\cdot\kappa+\overline{b}) - p_{i,j}(\kappa) + p_{i,j}(\kappa) - (\underline{a}\cdot\kappa+\underline{b})\Big)\, d\kappa \\ &= \int_{\underline{\kappa}}^{\overline{\kappa}} \overline{r}(\kappa;\overline{a},\overline{b})\, d\kappa \;+\; \int_{\underline{\kappa}}^{\overline{\kappa}} \underline{r}(\kappa;\underline{a},\underline{b})\, d\kappa \end{aligned}$$

Independent optimization of the bounds $(\underline{a},\underline{b})$ and $(\overline{a},\overline{b})$ allows us to minimize the area by minimizing the integrals of $\overline{r}(\kappa;\overline{a},\overline{b})$ and $\underline{r}(\kappa;\underline{a},\underline{b})$ separately.

Numerical integration principles (such as Riemann sums) dictate that the discrete average residual is a direct estimator of this continuous integral:

$$\int_{\underline{\kappa}}^{\overline{\kappa}} \overline{r}(\kappa;\overline{a},\overline{b})\, d\kappa \;\approx\; (\overline{\kappa}-\underline{\kappa})\cdot\left(\frac{1}{P}\sum_{p=1}^{P}\overline{r}(\kappa_p;\overline{a},\overline{b})\right)$$

This approximation is only accurate for a sufficiently large number of sample points P. Since the domain width $(\overline{\kappa}-\underline{\kappa})$ is a strictly positive constant, minimizing the discrete average residual for the upper (and lower) bounds directly guarantees the minimization of the total estimated over-approximation area Singh et al. [2019].

To further illustrate why minimizing the average residual is preferable to minimizing the maximum residual, we present Example 13.

Example 13 (Average vs. maximum residual and over-approximation area)**.** *As illustrated in Figure 4.4, consider a pixel function sampled at three points:* $p_{i,j}(1) = 0$, $p_{i,j}(2) = 2$, *and* $p_{i,j}(4) = 0$. *Let us assume the lower bound* $\underline{a}\kappa + \underline{b} = 0$ *and compute the upper bound* $\overline{a}\kappa + \overline{b}$ *under two different objectives:*

- ***Minimizing the average residual:*** *This objective minimizes the sum of the residuals, resulting in an upper bound* ($\overline{a}\kappa + \overline{b} = -\kappa + 4$) *that closely follows the global shape of the samples. This yields the smallest possible over-approximation area:*

$$Area = \int_1^4 ((-\kappa + 4) - 0)\, d\kappa = \mathbf{4.5} \qquad \max_p \overline{r}(\kappa_p; \overline{a}, \overline{b}) = 3.0$$

- ***Minimizing the maximum residual:*** *This objective minimizes the worst-case residual. This results in the bound* ($\overline{a}\kappa + \overline{b} = 2$). *While the maximum residual is reduced, the total over-approximation area increases:*

$$Area = \int_1^4 (2 - 0)\, d\kappa = 6.0 \qquad \max_p \overline{r}(\kappa_p; \overline{a}, \overline{b}) = \mathbf{2.0}$$

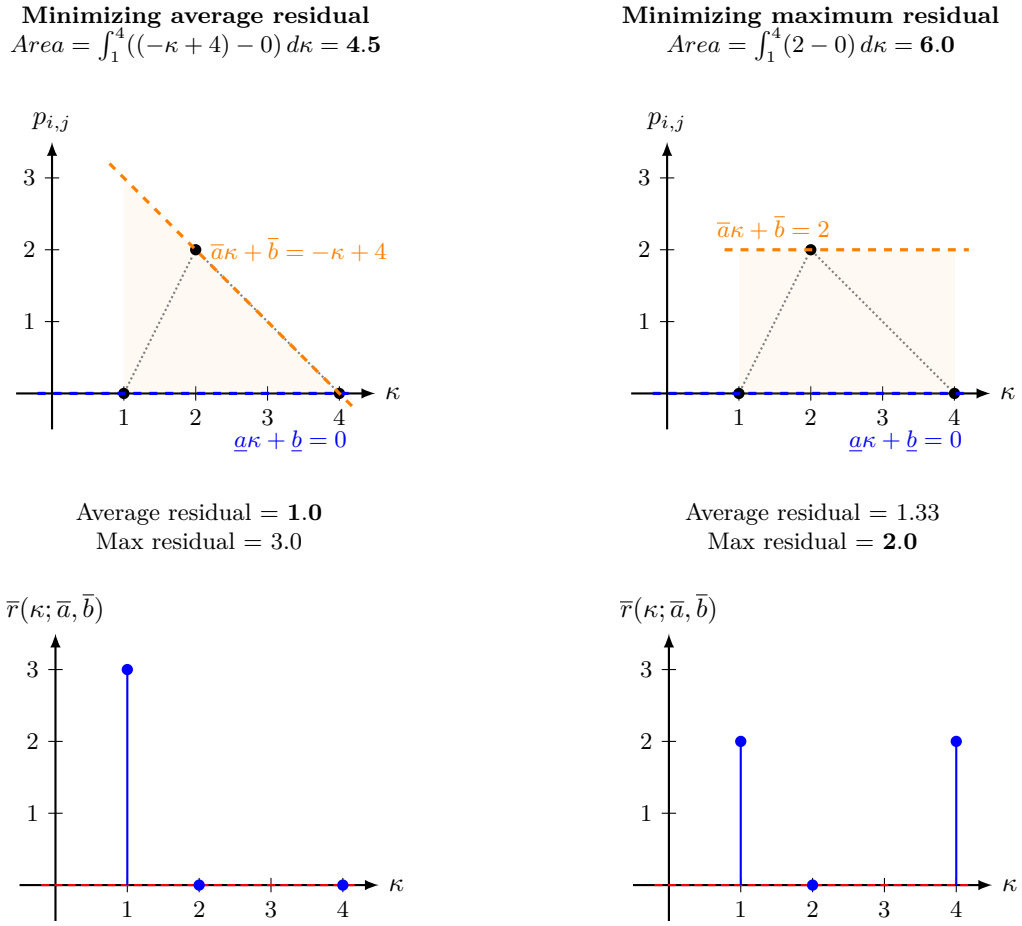


Figure 4.4: Illustration of Property 4.1. Minimizing the average residual (left) creates a smaller over-approximation area (4.5) than minimizing the maximum residual (right).

Consequently, we can translate the two requirements, validity and tightness, into the following optimization problem, denoted as $(\mathfrak{P})$:

$$\begin{array}{lll} \textit{Tightness} & \min\limits_{\overline{a},\overline{b}} & \left\{ \dfrac{1}{P} \displaystyle\sum_{p=1}^{P} \overline{r}(\kappa_p; \overline{a}, \overline{b}) \right\} \\ \textit{Validity} & \text{subject to} & \overline{r}(\kappa_p; \overline{a}, \overline{b}) \geq 0, \quad \forall p = 1, \ldots, P \end{array} \tag{$\mathfrak{P}$}$$

By solving this problem, we identify the optimal **unsound** linear bound, the tightest one that respects all sample constraints. The subsequent section addresses how DeepG and Super-DeepG achieve this.

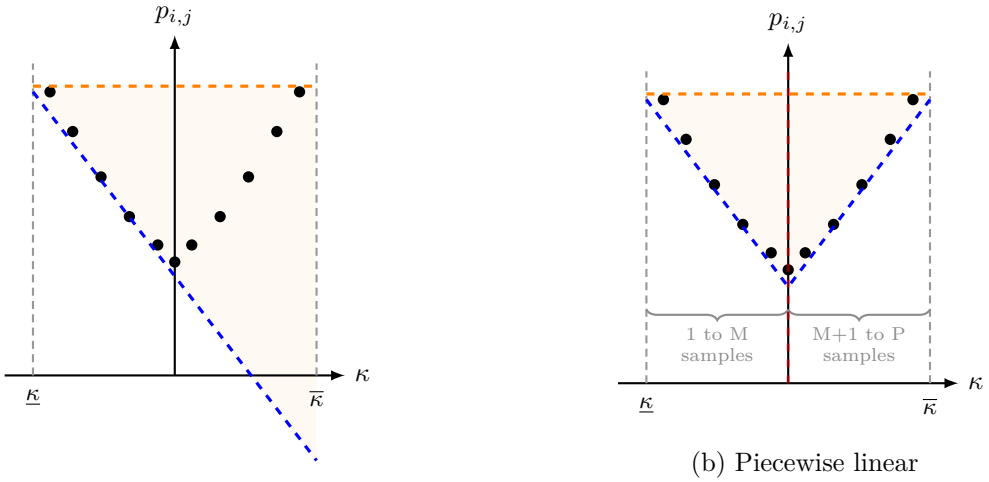
4.1.2 STEP 1.B: DeepG and PWL resolution

In practice, DeepG solves problem $(\mathfrak{P})$ directly as a Linear Program (LP) using Gurobi. While straightforward, this approach is computationally expensive.

PWL also employs the Gurobi solver, but on a different optimization problem, defined in $(\mathfrak{P}_{PWL})$. Instead of finding a single linear bound with coefficients $(\overline{a}, \overline{b})$, it seeks a piecewise linear bound made of two segments, $(\overline{a}_1, \overline{b}_1)$ and $(\overline{a}_2, \overline{b}_2)$. To achieve this, PWL splits the P samples into two equal halves of size $M = P/2$. Each segment is optimized to fit its own half as tightly as possible. However, **both** segments must remain valid bounds over the **entire** set of P samples. This leads to the following formulation:

$$\begin{array}{lll} \textit{Tightness} & \min\limits_{\overline{a}_1,\overline{b}_1} \left\{ \displaystyle\sum_{p=1}^{M} r(\kappa_p; \overline{a}_1, \overline{b}_1) \right\} & \min\limits_{\overline{a}_2,\overline{b}_2} \left\{ \displaystyle\sum_{p=M+1}^{P} r(\kappa_p; \overline{a}_2, \overline{b}_2) \right\} \\ \textit{Validity} & \text{subject to} \quad r(\kappa_p; \overline{a}_1, \overline{b}_1) \geq 0 \quad \forall p = 1, \ldots, P & \\ & \qquad\qquad\quad r(\kappa_p; \overline{a}_2, \overline{b}_2) \geq 0 \quad \forall p = 1, \ldots, P & \end{array} \tag{$\mathfrak{P}_{PWL}$}$$

Figure 4.5 (a) illustrates DeepG using $P = 10$ samples, and Figure 4.5 (b) illustrates PWL with $M = P/2 = 5$.



(a) Single linear over-approx. solving $(\mathfrak{P})$ (DeepG)

(b) Piecewise linear over-approx. solving $(\mathfrak{P}_{PWL})$ (PWL)

Figure 4.5: Comparison of the over-approximations computed by $(\mathfrak{P})$ (DeepG), $(\mathfrak{P}_{PWL})$ (PWL).

Rather than enforcing a piecewise bound over the entire domain, we can imagine dividing the sample points in half and apply DeepG's method twice, as represented in Figure 4.6. Consider another optimization problem defined in ($\mathfrak{P}_{split}$). This is equivalent to solving DeepG's problem ($\mathfrak{P}$) independently for each subset of samples (indices $1, \ldots, M$ and $M+1, \ldots, P$).

$$\begin{aligned} &\textit{Tightness} && \min_{\overline{a_1},\overline{b_1}} \left\{ \sum_{p=1}^{M} r(\kappa_p; \overline{a}_1, \overline{b}_1) \right\} \qquad \min_{\overline{a_2},\overline{b_2}} \left\{ \sum_{p=M+1}^{P} r(\kappa_p; \overline{a}_2, \overline{b}_2) \right\} && (\mathfrak{P}_{split}) \\ &\textit{Validity} && \text{subject to} \quad r(\kappa_p; \overline{a}_1, \overline{b}_1) \geq 0 \quad \forall p = 1, \ldots, M \\ & && \qquad\qquad\quad r(\kappa_p; \overline{a}_2, \overline{b}_2) \geq 0 \quad \forall p = M+1, \ldots, P \end{aligned}$$

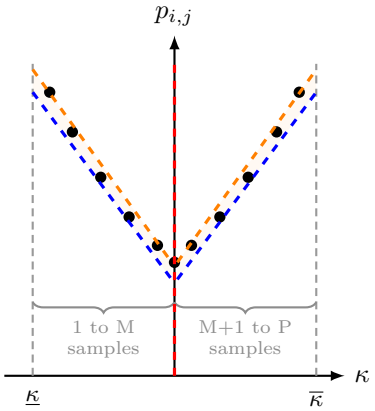


Figure 4.6: Two linear over-approximation solving ($\mathfrak{P}_{split}$) (DeepG on subsets).

Proposition 2 (Split vs. Piecewise linear)**.** *Solving the split optimization problem* ($\mathfrak{P}_{split}$) *always yields an over-approximation that is tighter than, or equal to, the piecewise linear problem* ($\mathfrak{P}_{PWL}$).

Proof 2. *Both problems minimize the exact same objective for tightness. However, they differ in their validity constraints. In the PWL problem* ($\mathfrak{P}_{PWL}$)*, each linear segment must remain valid across all P samples. In contrast, the split problem* ($\mathfrak{P}_{split}$) *only requires each segment to be valid for its own subset of M samples. Because* ($\mathfrak{P}_{split}$) *enforces fewer validity constraints, its search space is less restricted. Consequently, the optimization process can always find bounds that are at least as tight; if not tighter. In Figure 4.5: the linear bounds (c) yield a tighter over-approximation than the piecewise linear bounds (b).*

Our method Super-DeepG uses linear bounds (as used in DeepG) instead of piecewise linear bounds (PWL) for two key reasons. First, relying on linear bounds avoids the high computational complexity of solving a MILP needed by PWL. Second, as demonstrated in Proposition 2, independently solving the linear sub-problems guarantees an over-approximation that is tighter than, or equal to, the global PWL approach. Therefore, the objective of Super-DeepG is to effectively and efficiently resolve the split optimization problem defined in ($\mathfrak{P}_{split}$).

Furthermore, solving ($\mathfrak{P}_{split}$) is mathematically equivalent to solving ($\mathfrak{P}$) twice: independently on smaller subsets of samples. The Super-DeepG challenge reduces to solving a single instance of ($\mathfrak{P}$) as fast as possible.

4.1.3 STEP 1.C: Super-DeepG resolution

Like PWL and DeepG, Super-DeepG begins by sampling P points $\{p_{i,j}(\kappa_p)\}_P$ within the input domain $[\underline{\kappa}, \overline{\kappa}]$. The goal is to **efficiently** compute the unsound linear bounds defined by $\overline{a}$ and $\overline{b}$ to solve the optimization problem $(\mathfrak{P})$.

To achieve this, Super-DeepG reformulates $(\mathfrak{P})$ and solves it using an analytical solution. This formulation allows for parallelization on GPUs, reducing computation time. The methodology proceeds as follows:

- First, we prove that any optimal solution to problem $(\mathfrak{P})$ must touch exactly $d+1$ sample points, where d is the dimension of the input domain (we remind that $d = 1$ for transformations such as rotation, scaling, and shearing and $d = 2$ for translation on the x-axis and y-axis). Solving the optimization problem $(\mathfrak{P})$ is therefore equivalent to solving the $\binom{P}{d+1}$ independent sub-problems and selecting the best solution among them.
- Consequently, we only need to evaluate a finite set of $\binom{P}{d+1}$ candidate bounds. For each candidate, we efficiently evaluate our two criteria:
  - **Validity:** We discard any upper bound candidate that does not remain above all P sample points.
  - **Tightness:** For the remaining valid candidates, we compute the average residual and select the one with the minimum value.

For one-dimensional problems ($d = 1$), we can reformulate the problem without testing all $\binom{P}{2}$ candidate pairs. By anchoring the linear bound to a single sample point, we can analytically calculate the **optimal** slope, defined by a second point, that satisfies the validity and tightness requirements. Indeed, we define P sub-problems $\mathfrak{P}^{m,1d}$. We detail the analytical resolution for 1D inputs before generalizing the algorithm to d-dimensional spaces (which requires testing all $\binom{P}{d}$ candidate pairs).

4.1.3.1 STEP 1.C: Background

Our approach relies on the following mathematical property: any optimal affine bound necessarily intersects exactly $d+1$ affinely independent sample points. We first establish this property.

Definition 4.1 (Affinely independent)**.** *A set of $d+1$ points $\{\kappa_1, \ldots, \kappa_{d+1}\}$ in $\mathbb{R}^d$ is affinely independent if the d vectors $\{\kappa_2 - \kappa_1, \ldots, \kappa_{d+1} - \kappa_1\}$ are linearly independent. Geometrically, this means the points do not lie in any affine subspace of dimension strictly less than d. For instance, in 1D, two points must be distinct from each other to define a segment. In 2D, three points must be non-collinear to form a triangle. Similarly, in 3D, four points must be non-coplanar to define a tetrahedron.*

Remark 35 (Practical affine independence)**.** *Achieving affine independence in practice is straightforward when generating samples:*

- ***In 1D ($d = 1$):*** *We only need $d+1 = 2$ points. If we generate a set of P distinct sample points, any pair chosen from this set is automatically affinely independent.*
- ***In multi-D ($d > 1$):*** *Generating affinely independent points can be done using random sampling. To understand why, consider the $d = 2$ case illustrated in Figure 4.7, which requires $d+1 = 3$ points. For these points to be affinely dependent, they need to align on a 1D straight line (Figure 4.7, left). The probability of 3 continuous **random** points falling exactly on a line with zero thickness in a 2D space is close*

to zero Billingsley [1995]. Instead, they form a 2D triangle (Figure 4.7, right). Therefore, under continuous random *sampling, affine independence holds.*

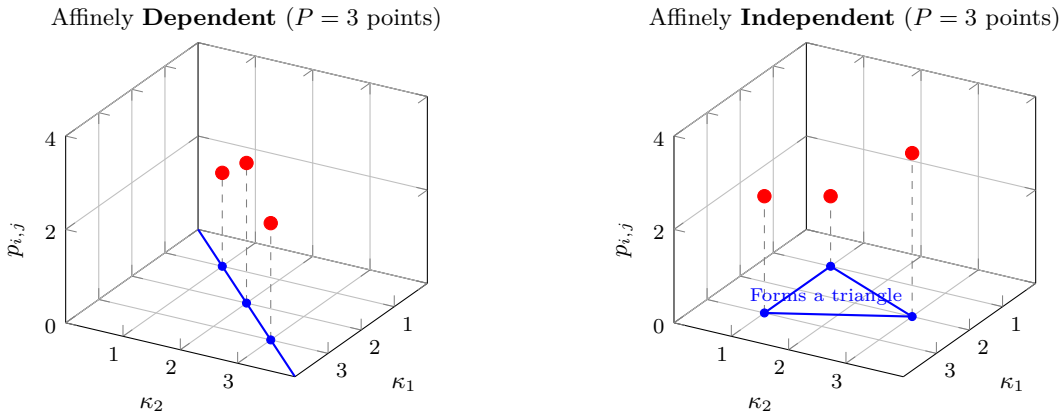


Figure 4.7: Visualization of affine independence for $d = 2$ (requiring $d+1 = 3$ points). The red dots are the 3D sample points $(\kappa, p_{i,j}(\kappa))$. The blue dots on the "floor" ($p_{i,j} = 0$) are the parameter vectors κ.

Proposition 3. *The optimization problem* $(\mathfrak{P})$ *admits an optimal solution* $(\overline{a}, \overline{b})$ *such that there exists a subset of* $d+1$ *affinely independent sample indices* $\mathcal{J} = \{p_1, \ldots, p_{d+1}\} \subset \{1, \ldots, P\}$ *that verify:*

$$\overline{a} \cdot \kappa_p + \overline{b} = p_{i,j}(\kappa_p), \quad \forall p \in \mathcal{J} \tag{4.3}$$

Solving $(\mathfrak{P})$ *is equivalent to independently solving* $\binom{P}{d+1}$ *sub-problems and selecting the tightest valid bound among them.*

Proof 3. *Define the variable* $z := (\overline{a}, \overline{b}) \in \mathbb{R}^{d+1}$. *The validity constraint for sample* p *can be written as:* $\overline{a} \cdot \kappa_p + \overline{b} \geq p_{i,j}(\kappa_p), \ \forall p = 1, \ldots, P$. *Indeed, by grouping the variables* $\overline{a}$ *and* $\overline{b}$ *into a single vector* $z := (\overline{a}, \overline{b})$, *the constraints define a polyhedron of the form:* $\{z \in \mathbb{R}^{d+1} \mid Cz \geq e\}$ *where each row of* $C \in \mathbb{R}^{P \times (d+1)}$ *is given by* $(\kappa_p, 1)$, *and* $e \in \mathbb{R}^P$ *is a vector whose* p*-th component is defined as* $e_p = p_{i,j}(\kappa_p)$.

Any set of $d+1$ *linearly independent active constraints defines a unique basic solution of this polyhedron, at least one of which is optimal Hillier and Lieberman [2015]; Bertsimas and Tsitsiklis [1997]; Ashish Goel [2008]. These correspond to a subset of* $d+1$ *affinely independent sample points* κ_p *for which Equation* (4.3) *holds for some index set* $\mathcal{J} = \{p_1, \ldots, p_{d+1}\} \subset \{1, \ldots, P\}$, *with* $|\mathcal{J}| = d+1$ *and the* $\{\kappa_p\}_{p \in \mathcal{J}}$ *affinely independent.*

Remark 36 (P can be low)**.** *In practice, this enumeration is not a computational bottleneck, as experiments confirm that a very small number of sample points* P *is sufficient (see Figure 4.24 in the experimental evaluation Section 4.4), while the dimension* d *of the geometric transformations is low.*

4.1.3.2 STEP 1.C: Super-DeepG resolution - Mathematically 1-dimension

When $\kappa \in \mathbb{R}$, the methodology proceeds as follows:

- **Decomposition into sub-problems** $(\mathfrak{P}^{m,1d})$**:** We have proved that the optimal unsound linear bound must pass through 2 points. According to Proposition 3, we fix 1 point (κ_m, p_m) to act as a pivot, allowing us to analytically calculate the **optimal** slope, defined by a second point, that satisfies the validity and tightness

requirements. Solving $(\mathfrak{P})$ is equivalent to solving the P independent sub-problems $(\mathfrak{P}^{m,1d})$ and selecting the best solution among them. This limits our search to exactly P candidate sub-problems.

- **Analytical solution:** For each candidate, we analytically evaluate our two criteria:
  - **Validity:** We keep only the bounds that safely remain above all P sample points.
  - **Tightness:** Among those valid bounds, we analytically select the tightest slope using the $P-1$ remaining points.
- **Global solution:** Finally, we compare the results from all P sub-problems and select the best bound.

Decomposition into sub-problems. We define a sub-problem $(\mathfrak{P}^{m,1d})$ by constraining the line to pass exactly through the sample point at κ_m:

$$\begin{aligned} &\textit{Tightness} && \min_{\overline{a},\overline{b}} \quad \left\{ \frac{1}{P} \sum_{p=1}^{P} \overline{r}(\kappa_p; \overline{a}, \overline{b}) \right\} \\ &\textit{Validity} && \text{subject to} \quad \overline{r}(\kappa_p; \overline{a}, \overline{b}) \geq 0, \quad \forall p = 1, \dots, P \\ & && \qquad\quad \overline{r}(\kappa_m, \overline{a}, \overline{b}) = 0. \end{aligned} \tag{$\mathfrak{P}^{m,1d}$}$$

An example is illustrated in Figure 4.8 when $P = 5$ and the sub-problem $(\mathfrak{P}^m)$ is evaluated for the third sample point ($m = 3$). Here, the red node represents the pivot point p_m at κ_m, through which all candidate lines must pass. The black nodes represent the remaining samples. By fixing the line to intersect p_m, the optimization reduces to finding the optimal rotation (slope) that minimizes the area under the line while ensuring that no other sample point ends up above it.

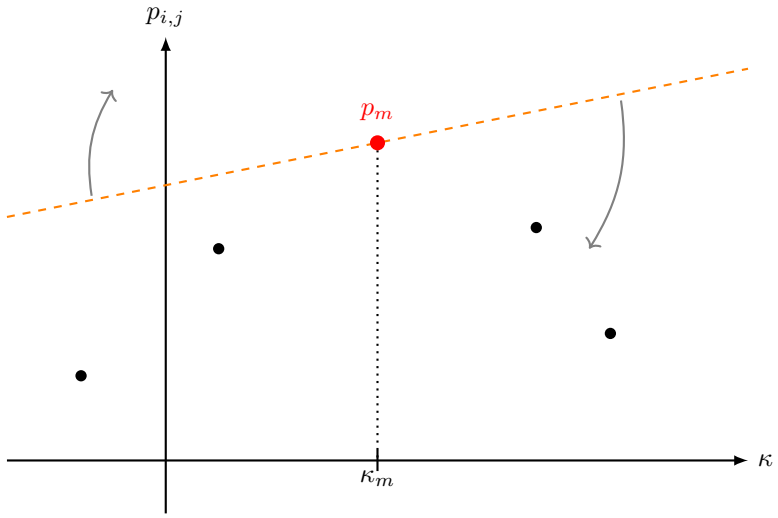


Figure 4.8: Visualization of a single sub-problem $(\mathfrak{P}^m)$.

Analytical solution of $(\mathfrak{P}^{m,1d})$. We can solve each sub-problem analytically. If the line passes through point m, the bias $\overline{b^m}$ is entirely determined by the slope $\overline{a^m}$:

$$\overline{r}(\kappa_m, \overline{a^m}, \overline{b^m}) = 0 \quad \Leftrightarrow \quad \overline{b^m} = p_{i,j}(\kappa_m) - \overline{a^m}\kappa_m. \tag{4.4}$$

Substituting this back into the definition of the residual, we see that the residual at any other point depends only on the slope $\overline{a^m}$:

$$\begin{aligned}\overline{r}(\kappa;\overline{a^m},\overline{b^m}) &= \overline{a^m}\cdot\kappa + \overline{b^m} - p_{i,j}(\kappa)\\ &= \overline{a^m}\cdot\kappa + p_{i,j}(\kappa_m) - \overline{a^m}\kappa_m - p_{i,j}(\kappa)\\ &= (p_{i,j}(\kappa_m) - p_{i,j}(\kappa)) - \overline{a^m}\cdot(\kappa_m - \kappa)\end{aligned} \tag{4.5}$$

We define this single-variable residual as $\overline{r}_m(\kappa;\overline{a^m})$. The sub-problem $(\mathfrak{P}^{m,1d})$ simplifies to finding the optimal slope $\overline{a^m}$ that minimizes the average residual while keeping all residuals positive:

$$\begin{array}{lll} \textit{Tightness} & \min\limits_{\overline{a^m}} & \left\{\dfrac{1}{P}\displaystyle\sum_{p=1}^{P}\overline{r}_m(\kappa_p;\overline{a^m})\right\} \\ \textit{Validity} & \text{subject to} & \overline{r}_m(\kappa_p;\overline{a^m}) \geq 0 \quad \forall p = 1,\dots,P. \end{array} \tag{$\mathfrak{P}^{m,1d}$}$$

Validity To enforce the validity requirement for every sample point $p \in \{1,\dots,P\}$, we demand that the residual remains positive. Substituting the expression from Equation (4.5), we get:

$$\begin{aligned}&\overline{r}_m(\kappa_p;\overline{a^m}) \geq 0\\ \Leftrightarrow\ &p_{i,j}(\kappa_m) - p_{i,j}(\kappa_p) - \overline{a^m}\cdot(\kappa_m - \kappa_p) \geq 0\\ \Leftrightarrow\ &\overline{a^m}\cdot(\kappa_m - \kappa_p) \leq p_{i,j}(\kappa_m) - p_{i,j}(\kappa_p)\end{aligned} \tag{4.6}$$

Isolating the slope $\overline{a^m}$ from this inequality Equation (4.6) requires dividing by the distance $(\kappa_m - \kappa_p)$. Because dividing by a negative number flips the inequality direction, the valid range for our slope strictly depends on whether the sample point κ_p is located to the left or to the right of the pivot κ_m:

$$\begin{cases} \overline{a^m} \leq \frac{p_{i,j}(\kappa_m)-p_{i,j}(\kappa_p)}{\kappa_m-\kappa_p} & \text{if } \kappa_m - \kappa_p > 0 \\ \overline{a^m} \geq \frac{p_{i,j}(\kappa_m)-p_{i,j}(\kappa_p)}{\kappa_m-\kappa_p} & \text{if } \kappa_m - \kappa_p < 0 \end{cases} \tag{4.7}$$

We split the other sample points into two sets relative to our pivot κ_m:

$$\mathcal{M}^+ = \{p \mid \kappa_m - \kappa_p > 0\} \quad \text{and} \quad \mathcal{M}^- = \{p \mid \kappa_m - \kappa_p < 0\} \tag{4.8}$$

For the bound to remain above all points (validity constraint), the slope $\overline{a^m}$ must fall within a specific interval defined by the slopes connecting κ_m to the other points:

$$\overline{a^m} \in \left[\max_{p\in\mathcal{M}^-}\frac{p_{i,j}(\kappa_m) - p_{i,j}(\kappa_p)}{\kappa_m - \kappa_p},\ \min_{p\in\mathcal{M}^+}\frac{p_{i,j}(\kappa_m) - p_{i,j}(\kappa_p)}{\kappa_m - \kappa_p}\right]. \tag{4.9}$$

Figure 4.9 visualizes the valid interval for the candidate slope defined by Equation (4.9).

Property 4.2 (Infeasibility of $(\mathfrak{P}^{m,1d})$)**.** *For a candidate pivot point m, the optimization problem $(\mathfrak{P}^{m,1d})$ has no solution if the maximum slope from $\mathcal{M}^-$ is strictly greater than the minimum slope from $\mathcal{M}^+$:*

$$\max_{p\in\mathcal{M}^-}\frac{p_{i,j}(\kappa_m) - p_{i,j}(\kappa_p)}{\kappa_m - \kappa_p} \quad > \quad \min_{p\in\mathcal{M}^+}\frac{p_{i,j}(\kappa_m) - p_{i,j}(\kappa_p)}{\kappa_m - \kappa_p} \tag{4.10}$$

In this scenario, any upper bound passing through p_m necessarily violates the constraints of at least one other sample point. There is no slope that provides a valid solution. Therefore, we discard this candidate point m. Figure 4.10 illustrates this situation.

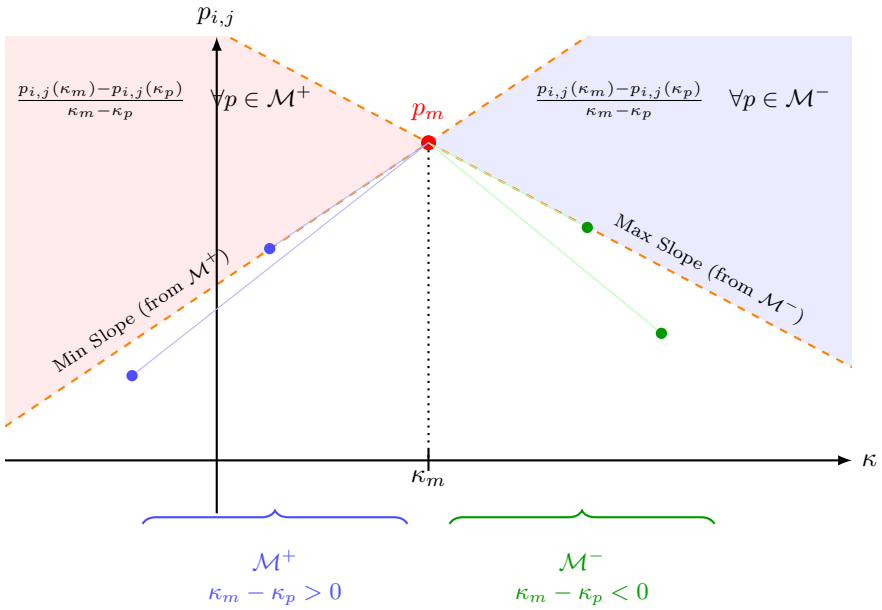


Figure 4.9: Visualisation of the valid slope interval. The slope is bounded above by the tightest point in $\mathcal{M}^+$ and below by the tightest point in $\mathcal{M}^-$.

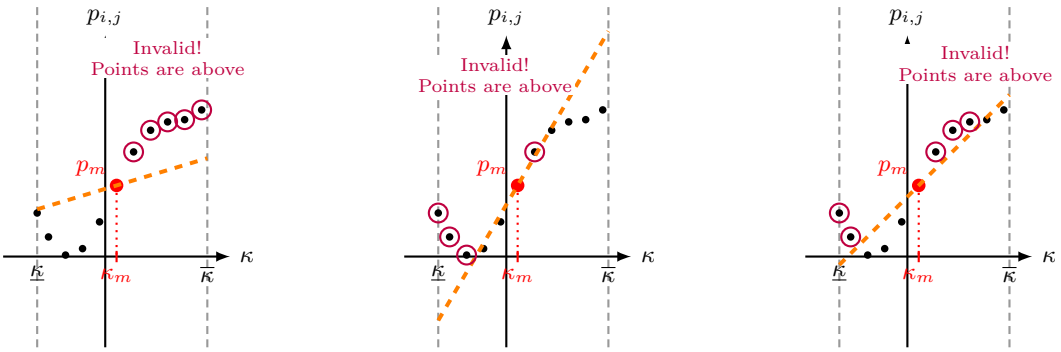


Figure 4.10: Illustration of an empty slope interval. Selecting the sample point p_m at κ_m as a pivot makes it impossible to build a valid upper bound.

Although Property 4.2 demonstrates that certain pivot choices result in an empty slope interval and must be discarded, we can formally guarantee that the optimization process always yields at least one valid candidate.

Proposition 4 (Existence of a feasible sub-problem)**.** *Given a finite set of P sample points, there exists at least one pivot index $m \in \{1, \dots, P\}$ for which the sub-problem $(\mathfrak{P}^{m,1d})$ is feasible. Consequently, the decomposition into sub-problems is guaranteed to yield at least one valid global linear over-approximation.*

Proof 4. *Let m^* be the index of the sample point with the maximum value, such that $p_{i,j}(\kappa_{m^*}) \geq p_{i,j}(\kappa_p)$ for all $p \in \{1, \dots, P\}$. We evaluate the feasibility of the sub-problem $(\mathfrak{P}^{m,1d})$ for this specific pivot, denoted as $\mathfrak{P}^{m^*,1d}$.*

According to Property 4.2, the sub-problem is feasible if and only if its valid slope interval is non-empty, which requires:

$$\max_{p \in \mathcal{M}^-} \frac{p_{i,j}(\kappa_{m^*}) - p_{i,j}(\kappa_p)}{\kappa_{m^*} - \kappa_p} \leq \min_{p \in \mathcal{M}^+} \frac{p_{i,j}(\kappa_{m^*}) - p_{i,j}(\kappa_p)}{\kappa_{m^*} - \kappa_p}$$

Because m^* corresponds to the global maximum of the sample set, the numerator

$(p_{i,j}(\kappa_{m^*}) - p_{i,j}(\kappa_p))$ is strictly positive (≥ 0) for all p. We can then analyze the sign of the bounding slopes:

- *Left part of the inequality - When $p \in \mathcal{M}^-$, the denominator $(\kappa_{m^*} - \kappa_p)$ is strictly negative. Thus, the resulting ratio is always ≤ 0.*
- *Right part of the inequality - When $p \in \mathcal{M}^+$, the denominator $(\kappa_{m^*} - \kappa_p)$ is strictly positive. Thus, the resulting ratio is always ≥ 0.*

The inequality holds. The valid slope interval always safely contains at least the value 0. Therefore, the sub-problem defined by m^* is feasible, guaranteeing at least one valid solution to the optimization problem.

At this stage, the bound is only anchored to the single pivot point p_m, yielding the continuous interval of valid slopes defined by Equation (4.9). The second anchoring point is determined by the tightness objective.

Tightness The objective function (average residual) is linear with respect to $\overline{a^m}$:

$$\frac{1}{P}\sum_{p=1}^{P}\overline{r}_m(\kappa_p;\overline{a^m}) = \left(\frac{1}{P}\sum_{p=1}^{P}\Big(p_{i,j}(\kappa_m) - p_{i,j}(\kappa_p)\Big)\right) - \overline{a^m}\left(\kappa_m - \frac{1}{P}\sum_{p=1}^{P}\kappa_p\right) \tag{4.11}$$

The minimum occurs at one of the boundaries of the interval of Equation (4.9). The direction of minimization depends on the distribution of sample points relative to the pivot. The optimal slope is given by:

$$\overline{a^{opt,m}} = \begin{cases} \min\limits_{p\in\mathcal{M}^+} \dfrac{p_{i,j}(\kappa_m) - p_{i,j}(\kappa_p)}{\kappa_m - \kappa_p} & \text{if } \left(\kappa^m - \sum\limits_{p=1}^{P}\frac{\kappa^p}{P}\right) \geq 0, \\ \max\limits_{p\in\mathcal{M}^-} \dfrac{p_{i,j}(\kappa_m) - p_{i,j}(\kappa_p)}{\kappa_m - \kappa_p} & \text{otherwise.} \end{cases} \tag{4.12}$$

With the optimal slope established, the corresponding bias is determined (the bound must intersect the pivot point at κ_m):

$$\overline{b^{opt,m}} = p_{i,j}(\kappa_m) - \overline{a^{opt,m}}\kappa_m \tag{4.13}$$

To compare this candidate bound against those generated by other pivot points, we evaluate its tightness by computing the average residual over all P samples:

$$\begin{aligned} \text{AvgRes}(\overline{a^{opt,m}}, \overline{b^{opt,m}}) &= \frac{1}{P}\sum_{p=1}^{P}\overline{r}(\kappa_p;\overline{a^{opt,m}},\overline{b^{opt,m}}) \\ &= \left[\overline{a^{opt,m}}\left(\frac{1}{P}\sum_{p=1}^{P}\kappa_p\right) + \overline{b^{opt,m}} - \left(\frac{1}{P}\sum_{p=1}^{P}p_{i,j}(\kappa_p)\right)\right] \end{aligned} \tag{4.14}$$

4.1.3.2.1 Global Solution. Finally, Super-DeepG computes the optimal parameters $(\overline{a^{opt}}, \overline{b^{opt}})$ by comparing the solutions of all sub-problems:

$$(\overline{a^{opt}}, \overline{b^{opt}}) = arg\min_{m=1,\dots,P} \text{AvgRes}(\overline{a^{opt,m}}, \overline{b^{opt,m}})$$

In summary, Super-DeepG enumerates all scenarios where the linear relaxation *touches* at least one sample point of the pixel function $p_{i,j}$. By solving these sub-problems and selecting the minimum, we obtain the exact same relaxation coefficients as DeepG's Step 1.

4.1.3.3 STEP 1.C: Super-DeepG resolution - Algorithm - 1-dimension

To run our mathematical formulation efficiently on a GPU, we translate the sub-problem decomposition into a tensor-based algorithm. The evaluation of every candidate pivot point m is entirely independent of the others. This allows the GPU to compute the slopes, validate the constraints, and calculate the residuals **for all** P **points** in a batched operation. In practice, as explained in Subsection 4.1.3.4 it is furthermore done across thousands of pixels simultaneously in a batched operation.

Inputs: $\kappa[.] \in \mathbb{R}^P, \quad p_{i,j}[.] \in \mathbb{R}^P$

> $p_{i,j}[.]$ contains the corresponding P pixel values and $\kappa[.]$ contains the associated P transformation parameters κ within the range of perturbations $[\underline{\kappa}, \overline{\kappa}]$.

Step A: Build matrix $K \in \mathbb{R}^{P\times P}$ and $A \in \mathbb{R}^{P\times P}$:

$$K[p,m] = \begin{cases} \kappa[p] - \kappa[m] & \text{if } p \neq m \\ 1 & \text{otherwise} \end{cases} \qquad A[p,m] = \frac{p_{i,j}[p] - p_{i,j}[m]}{K[p,m]}$$

> First, we compute all possible slopes between every pair of sample points. This prepares us to find the boundaries of the valid slope intervals required by Equation (4.9).
>
> *Note 1: These matrices have a dimension of $P \times P$ because we are evaluating every possible candidate pivot point (P candidates) against every sample point (P points) simultaneously in a batched operation.*
>
> *Note 2: We set $K[p,m] = 1$ when a point is paired with itself to safely prevent dividing by zero.*

Step B: Build matrices $A^{\mathcal{M}^+} \in \mathbb{R}^{P\times P}$ and $A^{\mathcal{M}^-} \in \mathbb{R}^{P\times P}$:

$$A^{\mathcal{M}^+}[p,m] = \begin{cases} A[p,m] & \text{if } K[p,m] < 0 \text{ and } p \neq m \\ +\infty & \text{otherwise.} \end{cases}$$

$$A^{\mathcal{M}^-}[p,m] = \begin{cases} A[p,m] & \text{if } K[p,m] > 0 \text{ and } p \neq m \\ -\infty & \text{otherwise.} \end{cases}$$

> Next, we split the slopes into two groups for each pivot point. One group handles points to the left ($\mathcal{M}^+$), the other handles points to the right ($\mathcal{M}^-$) as in Equation (4.8).
>
> *Note: We push infeasible solutions (Property 4.2) to infinity so they do not break our minimum and maximum calculations later.*

Step C: Build vector $A^{\mathcal{M}^-} \in \mathbb{R}^P$ and $A^{\mathcal{M}^+} \in \mathbb{R}^P$ from matrices

$$A^{\mathcal{M}^-}[m] = \max_p A^{\mathcal{M}^-}[p,m], \quad A^{\mathcal{M}^+}[m] = \min_p A^{\mathcal{M}^+}[p,m]$$

Now, we find the tightest slope constraints for each pivot point. This computes the lower and upper boundaries of the valid slope interval defined in Equation (4.9).

Step D: Build vector $A^{\mathcal{M}^-} \in \mathbb{R}^I$ and $A^{\mathcal{M}^+} \in \mathbb{R}^J$ where $I \leq P$ and $J \leq P$

$$\text{cond}_{\text{empty}} = \left(A^{\mathcal{M}^-}[m] \leq A^{\mathcal{M}^+}[m]\right)$$

Letting $\mathcal{M}$ denote either $\mathcal{M}^-$ or $\mathcal{M}^+$:

$$mask^{\mathcal{M}}[m] = \begin{cases} True & \text{if cond}_{\text{empty}} \text{ and } A^{\mathcal{M}}[m] \neq \pm\infty \\ False & \text{otherwise} \end{cases}$$

$$A^{\mathcal{M}}[.] = A^{\mathcal{M}}[mask^{\mathcal{M}}], \qquad p_{i,j}^{\mathcal{M}}[.] = p_{i,j}[mask^{\mathcal{M}}] \qquad \kappa^{\mathcal{M}}[.] = \kappa[mask^{\mathcal{M}}]$$

As explained in Remark 4.2, some pivot points cannot form a valid upper bound. The optimization problem has no solution. We check if the slope interval is empty using the logic from Equation (4.10).

Remark 37 (Parallel computation of pivots)**.** *By structuring the data into $P \times P$ matrices in Steps A and B, we avoid using loops to check each candidate. When we compute the maximum and minimum in Step C and mask filtering in Step D, we find the valid slope boundaries for all P candidate pivots. As a result, the vectors $A^{\mathcal{M}^-}$ and $A^{\mathcal{M}^+}$ contain the tightest slope constraints for* ***every pivot simultaneously****.*

Step E: Computes the bias $B^{\mathcal{M}^-} \in \mathbb{R}^I$ and $B^{\mathcal{M}^+} \in \mathbb{R}^J$

$$B^{\mathcal{M}^-}[i] = p_{i,j}^{\mathcal{M}^-}[i] - A^{\mathcal{M}^-}[i] \cdot \kappa^{\mathcal{M}^-}[i] \qquad B^{\mathcal{M}^+}[j] = p_{i,j}^{\mathcal{M}^+}[j] - A^{\mathcal{M}^+}[j] \cdot \kappa^{\mathcal{M}^+}[j]$$

We calculate the matching bias as Equation (4.13).

Step F: Final selection

$$\text{average}_{\text{residual}}(x, y) = x \cdot \kappa^{\text{mean}} + y \quad - \quad p_{i,j}^{\text{mean}} \qquad \text{with} \qquad \begin{cases} \kappa^{\text{mean}} & = & \frac{1}{P} \sum_{k=1}^{P} \kappa[k] \\ p_{i,j}^{\text{mean}} & = & \frac{1}{P} \sum_{k=1}^{P} p_{i,j}[k] \end{cases}$$

Letting $\mathcal{M}$ denote either $\mathcal{M}^-$ or $\mathcal{M}^+$:

$$(\overline{a^{\mathcal{M}}}, \overline{b^{\mathcal{M}}}) \;=\; \arg\min_i \left\{ \text{average}_{\text{residual}}(A^{\mathcal{M}}[i], B^{\mathcal{M}}[i]) \right\}$$

As shown in Equation (4.14), we calculate the average residual for each slope and bias pair. We then isolate the specific candidate that gives the lowest average residual in both groups.

Step G: Return result

$$(\overline{a^{opt}}, \overline{b^{opt}}) \;=\; \arg\min_{\mathcal{M}^- \mathcal{M}^+} \left\{ \text{average}_{\text{residual}}(\overline{a^{\mathcal{M}}}, \overline{b^{\mathcal{M}}}) \right\}$$

Return $\left(\overline{a^{opt}}, \overline{b^{opt}}\right)$

> Finally, we compare the best candidate from the left-side bounds with the best from the right-side bounds. We pick the one with the absolute smallest average residual. This returns the tightest unsound linear bound.

As shown in the algorithm, every candidate pivot point m is evaluated simultaneously in parallel (see Remark 37). We now introduce further optimizations to improve computational efficiency using PyTorch.

4.1.3.4 STEP 1.C: Super-DeepG resolution - Code - 1-dimension

To execute the mathematical logic from Step 1.C.1d efficiently on a GPU, we translate the sub-problem decomposition into a highly parallelized tensor-based algorithm using PyTorch. The power of this implementation lies in its ability to leverage matrix operations to solve sub-problems simultaneously. Here are the key implementation tricks that make this algorithm performant:

4.1.3.4.1 Parallelization across all pixels Instead of computing the upper and lower bounds for one pixel at a time, the algorithm processes the entire image at once. The spatial dimensions of the image (Channels C, Height H, Width W) are preserved throughout the tensor operations. This means that solving ($\mathfrak{P}^{m,1d}$) is done concurrently for every single pixel in the $C \times H \times W$ grid.

4.1.3.4.2 Batch processing of multiple sub-intervals We further parallelize the search by evaluating multiple domain sub-intervals at the same time. For instance, if we split the continuous domain $[\underline{\kappa}, \overline{\kappa}]$ into 3 distinct sub-intervals, we sample P points in each. These intervals are stacked in a batch dimension *batch*. Consequently, our tensors operate with a 5-dimensional shape: `[batch, P, C, H, W]`. This allows the GPU to compute bounds for all sub-intervals and all pixels in few pass.

4.1.3.4.3 Broadcasting for elegant memory management (Step A) When calculating the slopes $A[p, m]$ between the samples, we divide the pixel differences by the transformation parameter differences K. Because K (the difference in κ) of shape `[batch, P]` is identical for all pixels, we do not need to duplicate it in memory. By reshaping K to `[batch, P, 1, 1, 1]`, PyTorch automatically "broadcasts" it across the image dimensions `[C, H, W]` during the division, saving memory.

```
# Step A
# K represents the differences in transformation param. (kappa_p - kappa_m)
# Shape of K: [batch, P, 1, 1, 1]
# pixel_diff represents the differences in pixel values (p_p - p_m)
# Shape of pixel_diff: [batch, P, C, H, W]

# PyTorch automatically broadcasts K across C, H, and W:
A_matrix = pixel_diff / K  # Output shape: [batch, P, C, H, W]
```

Listing 4.1: PyTorch implementation of Step A showing the broadcasting.

4.1.3.4.4 Branchless logic with `torch.where` (Steps B and C) In the mathematical formulation, dividing the points into the left set $\mathcal{M}^+$ and the right set $\mathcal{M}^-$ requires conditional logic (if $K < 0$, etc.). On a GPU, "if/else" statements create branching, which slows down execution. We implement this using `torch.where(condition, matrix, new_value)`, which returns `new_value` where the `condition` is true, and retains the original `matrix` values otherwise. We evaluate all conditions simultaneously and replace invalid slopes with $\pm\infty$. This allows us to use simple `.min()` and `.max()` reduction functions to find the tightest bounds.

```
# Step B: Filter slopes into set M+ (where K < 0).
# We replace invalid entries with +infinity.
A_M_plus = torch.where(K < 0, A_matrix, torch.tensor(float('inf')))

# Filter slopes into set M- (where K > 0).
# We replace invalid entries with -infinity.
A_M_minus = torch.where(K > 0, A_matrix, torch.tensor(float('-inf')))

# Step C: Reduce the P dimension to find the best constraint per pixel
# Output shapes transition from [batch, P, C, H, W] to [batch, C, H, W]
A_M_plus_bound = A_M_plus.min(dim=2).values
A_M_minus_bound = A_M_minus.max(dim=2).values
```

Listing 4.2: Simplified PyTorch implementation of Steps B and C using branchless tensor operations.

While the previous sections focused on the one-dimensional case where $\kappa \in \mathbb{R}$, we now generalize our approach to multi-dimensional geometric transformations where $\kappa \in \mathbb{R}^d$.

4.1.3.5 STEP 1.C: Super-DeepG resolution - Mathematically - multi-dimensions

In multi-dimensions, the input is a set of P sample points. Each point is defined by a parameter vector $\kappa \in \mathbb{R}^d$ and a pixel value $p_{i,j}(\kappa) \in \mathbb{R}$. For example, if $d = 2$, the parameter κ^1 can be the translation on the x-axis, and κ^2 is the translation on the y-axis.

Our goal is to compute the optimal coefficients for the linear bound. The output is a slope vector $\overline{a} \in \mathbb{R}^d$ and a single scalar bias $\overline{b} \in \mathbb{R}$.

As established in Proposition 3, we know that the optimal linear bound touch $d+1$ affinely independent sample points. Figure 4.11 illustrates the $d = 2$ case with $P = 7$, showing a candidate affine bound (a plane) defined by $d + 1 = 3$ affinely independent samples. The candidate is valid because all other samples (black points) remain strictly below the plane. Let us define a specific subset of $d+1$ sample points as $\mathcal{J} = \{p_1, \ldots, p_{d+1}\}$.

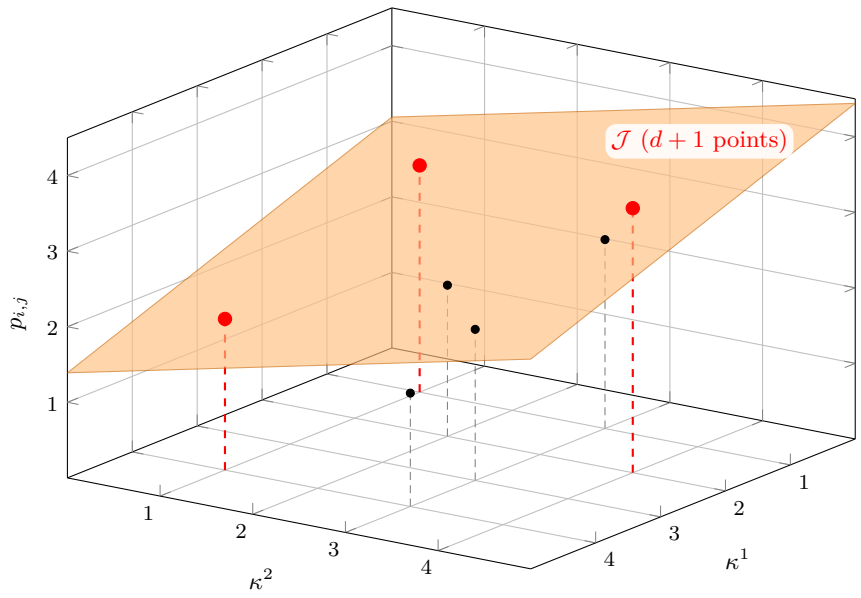


Figure 4.11: Visualization of the d-dimensional candidate enumeration for $d = 2$. A candidate upper bound (the orange plane) is defined by a subset of $d + 1 = 3$ affinely independent samples (red points).

Using this property, we can break down the main optimization problem ($\mathfrak{P}$) into

smaller, independent sub-problems. For every possible subset $\mathcal{J}$, we force our candidate hyperplane to pass through those $d+1$ points. We define this multi-dimensional sub-problem as follows:

$$\begin{array}{lll} \textit{Tightness} & \min\limits_{\overline{a},\overline{b}} & \left\{ \dfrac{1}{P} \displaystyle\sum_{p=1}^{P} \overline{r}(\kappa_p; \overline{a}, \overline{b}) \right\} \\ \textit{Validity} & \text{subject to} & \overline{r}(\kappa_p; \overline{a}, \overline{b}) \geq 0, \quad \forall p = 1, \dots, P \\ & & \overline{a} \cdot \kappa_p + \overline{b} = p_{i,j}(\kappa_p), \quad \forall p \in \mathcal{J} \end{array} \tag{$\mathfrak{P}^{\mathcal{J},multi-d}$}$$

In the sub-problem ($\mathfrak{P}^{\mathcal{J},multi-d}$), the parameters for the slope $\overline{a}$ and the bias $\overline{b}$ are fully determined by the equality in the validity constraints. Because there are no degrees of freedom left, the mathematical resolution flows in three straightforward steps:

- **Solve:** For a given subset $\mathcal{J}$, we calculate the unique candidate hyperplane $(\overline{a}, \overline{b})$ that passes exactly through those $d+1$ points.
- **Validity:** We evaluate whether the plane satisfies the constraints for the remaining samples (e.g., remaining strictly above all P samples for an upper bound). If the plane is invalid, we discard it.
- **Tightness:** For the valid candidates, we calculate the average residual. This allows us to compare subsets and ultimately select the tightest over-approximation.

Remark 38 (Multi-dimensional resolution applied to 1D)**.** *This method could be applied to the 1-dimensional problem ($d = 1$) by selecting subsets of $d + 1 = 2$ points, resulting in sub-problems $\binom{P}{2} = \frac{P \cdot (P-1)}{2}$ ($O(P^2)$). However, we retain the specific 1D formulation presented earlier. By fixing only one pivot point and analytically computing the optimal slope for the second point, we reduce the search space from $O(P^2)$ pairs down to just P sub-problems.*

4.1.3.6 STEP 1.C: Super-DeepG resolution - Algorithm - multi-dimensions

In this subsection, we detail the algorithmic steps used to implement the multi-dimensional mathematics on a GPU. For every subset of points, we solve a linear system to find the candidate coefficients, verify their validity against all samples, and select the tightest one.

By organizing these steps into a tensor-based algorithm, we move from a theoretical decomposition of sub-problems to a practical implementation that solves sub-intervals and pixels in parallel.

4.1.3.6.1 Select $d+1$ distinct parameters We select a subset of $d+1$ affinely independent parameter vectors $\kappa_1, \dots, \kappa_{d+1} \in \mathbb{R}^{d+1}$. We assume that the optimal affine upper bound passes through these points, i.e., they satisfy Equation (4.3).

4.1.3.6.2 Eliminate $\overline{b}$ Subtract the equation at $p = d+1$ from Equation (4.3) for $p = 1, \dots, d$:

$$\overline{a} \cdot (\kappa_p - \kappa_{d+1}) = p_{i,j}(\kappa_p) - p_{i,j}(\kappa_{d+1})$$

4.1.3.6.3 Solving the $d \times d$ system Collect the d equations obtained above into matrix form. Write $p = 1, \dots, d$ and $i = 1, \dots, d$, where the superscript i denotes the i-th component of the vector:

$$K_{pi} := (\kappa_p^i - \kappa_{d+1}^i) \in \mathbb{R}^{d \times d}$$

$$G_p := p_{i,j}(\kappa_p) - p_{i,j}(\kappa_{d+1}) \in \mathbb{R}^d$$

The problem reduces to a linear system where the unknown is the slope tensor $\overline{a} \in \mathbb{R}^d$: $K\overline{a} = G$. Because the $d+1$ selected points are affinely independent, the matrix K has full rank and is therefore invertible. We obtain a unique vector: $\overline{a} = K^{-1}G$. With $\overline{a}$ known, the bias is recovered from any point, for $k = 1$: $\overline{b} = p_{i,j}(\kappa_1) - \overline{a} \cdot \kappa_1$

4.1.3.6.4 Valid candidates If $\exists p \in 1, \ldots, P$ such that $\overline{r}(\kappa_p; \overline{a}, \overline{b}) \leq 0$. The chosen candidate must be discarded.

4.1.3.6.5 Enumerating candidates We repeat this procedure for all subsets of $d+1$ points among the P possible parameter combinations. For each subset, we retain the pair $(\overline{a}, \overline{b})$, and finally select the one that yields the lowest average residual.

4.1.3.7 STEP 1.C: Super-DeepG resolution - Code - multi-dimensions

To execute the multi-dimensional resolution efficiently, Super-DeepG discards the traditional use of iterative LP solvers (such as Gurobi) in favor of an analytical tensor implementation using PyTorch. We solve the linear system $K\overline{a} = G$ formulated in Paragraph 4.1.3.6.3 using the batched linear algebra operation `torch.linalg.solve`.

This direct resolution is possible because we construct our subsets using $d+1$ **affinely independent** parameter vectors (Remark 35). Consequently, the resulting $d \times d$ matrix K is mathematically guaranteed to be full rank and, therefore, strictly invertible.

LP solvers (such as Gurobi) are inherently CPU-bound and struggle to scale when solving thousands of small, independent optimization problems. PyTorch, however, excels at batched tensor operations. Because the matrix K is small ($d \times d$, e.g., 2×2 for 2D spatial transformations) and the number of P can be low (Remark 36), Super-DeepG stacks the matrices for all candidate subsets $\mathcal{J}$ across all pixels in the image into a single tensor. A single call to `torch.linalg.solve(K, G)` simultaneously computes the candidate bounds in parallel on the GPU.

4.1.4 STEP 1: Conclusion

In summary, as illustrated in Figure 4.12, the goal of Step 1 is to construct **unsound** linear bounds for the residual function. These bounds are only guaranteed to hold over a finite set of sample points. We evaluate our candidate bounds based on two simple rules: **validity** (staying strictly above or below all samples) and **tightness** (minimizing the average residual to reduce the over-approximation area).

Previous methods like DeepG and PWL rely on computationally expensive solvers to find these bounds. Furthermore, we demonstrated that solving linear sub-problems independently on split domains guarantees an over-approximation that is tighter than, or equal to, the globally constrained piecewise approach used by PWL.

To achieve this efficiently, Super-DeepG replaces the traditional solver with an analytical solution. This mathematical formulation translates directly into highly parallelized operations. As a result, we can evaluate and select the optimal linear bounds across pixels and sub-intervals simultaneously on a GPU.

4.2 STEP 2: Lipschitz constant

In Step 1, we construct the tightest possible linear bounds over a discrete set of samples. By shifting the lines appropriately (Step 3), we ensure they over-approximate

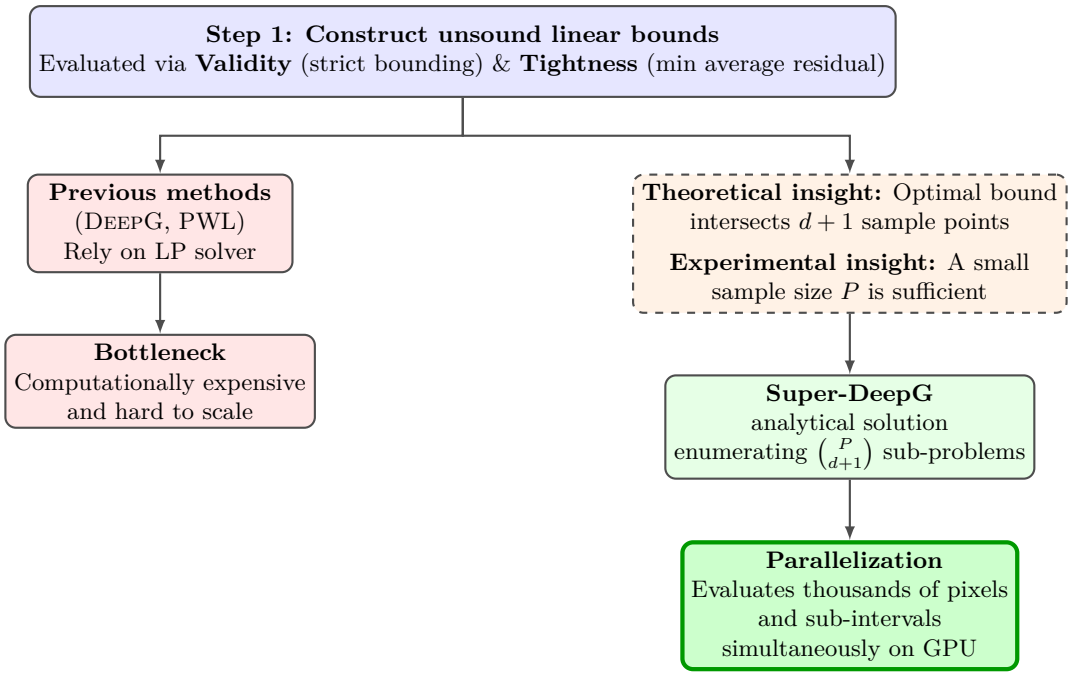


Figure 4.12: Overview of Step 1.

the true pixel function across the entire continuous domain, thus making them fully sound. To do so, the Lipschitz constant $\mathcal{L}_{i,j}$ for the residual is required.

Given an image x, the following calculation is performed independently for each pixel. To bound the worst-case variation of the residual $\overline{r}(\kappa; \overline{a}, \overline{b}) = (\overline{a} \cdot \kappa + \overline{b}) - p_{i,j}(\kappa)$ subject to geometric perturbations over the domain $[\underline{\kappa}, \overline{\kappa}]$, we determine its Lipschitz constant with respect to the parameters κ.

4.2.1 STEP 2.A: Problem statement

The tightest Lipschitz constant $\mathcal{L}^*_{i,j}$ is defined as the supremum of the gradient norm of the residual function Heinonen [2001]:

$$\mathcal{L}^*_{i,j} = \sup_{\kappa \in [\underline{\kappa}, \overline{\kappa}]} \left\| \nabla_\kappa \ \overline{r}(\kappa; \overline{a}, \overline{b}) \right\|_\infty \leq \sup_{\kappa \in [\underline{\kappa}, \overline{\kappa}]} \left\| \nabla_\kappa \ p_{i,j}(\kappa) \right\|_\infty + |\overline{a}| \tag{4.15}$$

Since evaluating the exact supremum over the range of perturbations $[\underline{\kappa}, \overline{\kappa}]$ is generally intractable, we instead compute a tractable upper bound $\mathcal{L}_{i,j} \geq \mathcal{L}^*_{i,j}$. This is achieved by bounding the gradient term $\nabla_\kappa \, p_{i,j}(\kappa)$.

This subsection details the step-by-step derivation of this bound. We begin with a reminder of the pixel variation, followed by an application of the chain rule to express the gradient. Finally, we analyze the derivatives of the interpolation function to establish an upper bound.

4.2.1.1 Decomposing the pixel variation

In Subsection 3.1.4, we introduced the inverse mapping and interpolation. Recall that the value of a perturbed pixel is obtained by mapping its integer coordinates (i, j) back to a real-valued location $(i', j') = T_\kappa^{-1}(i, j)$ in the original image, and then applying an interpolation function I_x. The pixel value is given by the composition:

$$p_{i,j}(\kappa) = I_x \circ T_\kappa^{-1}(i, j) = I_x(i', j')$$

To find the Lipschitz constant, we must compute the gradient $\nabla_\kappa\, p_{i,j}(\kappa)$. Since this is a composition of functions, we apply the chain rule.

4.2.1.2 Expression of the gradient of the pixel value

We denote the Jacobian matrix with respect to the parameters $\kappa = (\kappa_1, \ldots, \kappa_d \in \mathbb{R}^d)$ by:

$$J_{T_\kappa^{-1}}(\kappa) = \begin{pmatrix} \frac{\partial i'}{\partial \kappa_1} & \cdots & \frac{\partial i'}{\partial \kappa_d} \\ \frac{\partial j'}{\partial \kappa_1} & \cdots & \frac{\partial j'}{\partial \kappa_d} \end{pmatrix}$$

For fixed coordinates (i, j), the chain rule yields the following expression for the gradient:

$$\nabla_\kappa\, p_{i,j}(\kappa) \;=\; J_{T_\kappa^{-1}}(\kappa)^\top \times \nabla I_x\Big(T_\kappa^{-1}(i,j)\Big) \qquad \in \mathbb{R}^d$$

This matrix-vector product can be written component-wise. If we consider a single parameter component κ_l, the partial derivative is:

$$\frac{\partial p_{i,j}}{\partial \kappa_l} = \frac{\partial i'}{\partial \kappa_l}(\kappa) \cdot \frac{\partial I_x}{\partial i'}(i', j') \;+\; \frac{\partial j'}{\partial \kappa_l}(\kappa) \cdot \frac{\partial I_x}{\partial j'}(i', j') \tag{4.16}$$

Equation (4.16) highlights that the total variation depends on two factors: how the interpolation changes spatially ($\frac{\partial I_x}{\partial i'}, \frac{\partial I_x}{\partial j'}$), and how the coordinates change with respect to the parameters ($\frac{\partial i'}{\partial \kappa_l}, \frac{\partial j'}{\partial \kappa_l}$).

Example 14 (Lipschitz constant for rotation)**.** *Consider a rotation parameterized by $\kappa = \theta \in \mathbb{R}$, where θ denotes the angle of rotation around the origin. The inverse transformation using Table 3.2 from Subsection 3.1.2 is:*

$$T_\theta^{-1}(i,j) = \;\; (i',\; j') \;\; = (i\cos\theta + j\sin\theta,\; -i\sin\theta + j\cos\theta)^\top$$

Applying Equation (4.16) with respect to θ gives the following:

$$\begin{aligned} \frac{\partial p_{i,j}}{\partial \theta} &= \frac{\partial i'}{\partial \theta} \cdot \frac{\partial I_x}{\partial i'}(i', j') \;+\; \frac{\partial j'}{\partial \theta} \cdot \frac{\partial I_x}{\partial j'}(i', j') \\ \frac{\partial p_{i,j}}{\partial \theta} &= (-i\sin\theta + j\cos\theta) \cdot \frac{\partial I_x}{\partial i'}(i', j') \;+\; (-i\cos\theta - j\sin\theta) \cdot \frac{\partial I_x}{\partial j'}(i', j') \end{aligned} \tag{4.17}$$

Example 15 (Lipschitz constant for translation)**.** *Consider a translation parameterized by $\kappa = (t_1, t_2) \in \mathbb{R}^2$, where t_1 and t_2 denote shifts along the i- and j-axes. The inverse transformation is:*

$$T_{t_1,t_2}^{-1}(i,j) = \;\; (i',\; j') \;\; = \;\; (i - t_1, j - t_2)^\top$$

Applying Equation (4.16) with respect to t_1 gives the following:

$$\begin{aligned} \frac{\partial p_{i,j}}{\partial t_1} &= \frac{\partial i'}{\partial t_1} \cdot \frac{\partial I_x}{\partial i'}(i', j') \;+\; \frac{\partial j'}{\partial t_1} \cdot \frac{\partial I_x}{\partial j'}(i', j') \\ \frac{\partial p_{i,j}}{\partial t_1} &= (-1) \cdot \frac{\partial I_x}{\partial i'}(i', j') \;+\; (0) \cdot \frac{\partial I_x}{\partial j'}(i', j') \end{aligned} \tag{4.18}$$

Applying Equation (4.16) with respect to t_2 gives the following:

$$\begin{aligned} \frac{\partial p_{i,j}}{\partial t_2} &= \frac{\partial i'}{\partial t_2} \cdot \frac{\partial I_x}{\partial i'}(i', j') \;+\; \frac{\partial j'}{\partial t_2} \cdot \frac{\partial I_x}{\partial j'}(i', j') \\ \frac{\partial p_{i,j}}{\partial t_2} &= (0) \cdot \frac{\partial I_x}{\partial i'}(i', j') \;+\; (-1) \cdot \frac{\partial I_x}{\partial j'}(i', j') \end{aligned} \tag{4.19}$$

While computing the Jacobian terms $\frac{\partial i'}{\partial \kappa_l}$ and $\frac{\partial j'}{\partial \kappa_l}$ is straightforward, the interpolation derivatives require further explanation.

4.2.1.3 Bounding the interpolation gradient

First, we focus on bounding the derivatives of the interpolation function, $\frac{\partial I_x}{\partial i'}$ and $\frac{\partial I_x}{\partial j'}$. In Subsection 3.1.4 we defined:

- p_0, p_1, p_2, and p_3 are the pixel values of the four nearest integer neighbors around the transformed coordinate (i', j').
- α and β represent the fractional offsets of (i', j') from its bottom-left integer neighbor $p_0 = (i_0, j_0)$. Specifically, $\alpha = i' - i_0$ measures the horizontal offset, and $\beta = j' - j_0$ measures the vertical offset. Both α and β are bounded in the interval $[0, 1]$.

With these terms defined, the interpolated value is:

$$I_x(i', j') \;=\; (1-\alpha)\,(1-\beta)\,p_0 \;+\; (1-\alpha)\,\beta\,p_1 \;+\; \alpha\,(1-\beta)\,p_2 \;+\; \alpha\,\beta\,p_3 \tag{4.20}$$

By applying the chain rule to the interpolation formula, and noting that $\frac{\partial \alpha}{\partial i'} = 1$ (since the integer coordinate i_0 is constant within a specific grid cell, i.e. between four neighbors), we obtain:

$$\frac{\partial I_x(i', j')}{\partial i'} = \frac{\partial I_x(i, j)}{\partial \alpha} \cdot \frac{\partial \alpha}{\partial i'} = \frac{\partial I_x(i, j)}{\partial \alpha}$$

Differentiating the bilinear expression in Equation (4.20) gives us:

$$\frac{\partial I_x(i', j')}{\partial i'} = \frac{\partial I_x(i', j')}{\partial \alpha} = p_2 - p_0 + (-p_2 + p_0 + p_3 - p_1) \cdot \beta$$

Similarly, for the vertical direction:

$$\frac{\partial I_x(i', j')}{\partial j'} = \frac{\partial I_x(i', j')}{\partial \beta} = p_1 - p_0 + (-p_1 + p_0 + p_3 - p_2) \cdot \alpha$$

We observe that these partial derivatives are affine functions of β and α, respectively. Since an affine function on the unit interval $[0, 1]$ achieves its maximum absolute value at the endpoints (0 or 1):

$$\frac{\partial I_x(i', j')}{\partial \alpha} = \begin{cases} p_2 - p_0 & \text{if } \beta = 0 \\ p_3 - p_1 & \text{if } \beta = 1 \end{cases} \qquad \frac{\partial I_x(i', j')}{\partial \beta} = \begin{cases} p_1 - p_0 & \text{if } \alpha = 0 \\ p_3 - p_2 & \text{if } \alpha = 1 \end{cases} \tag{4.21}$$

It follows that:

$$\begin{aligned} \left|\frac{\partial I_x}{\partial i'}(i', j')\right| &\le \mathcal{A}^{(1)}_{i,j} = \max(|p_2 - p_0|, |p_3 - p_1|), \\ \left|\frac{\partial I_x}{\partial j'}(i', j')\right| &\le \mathcal{A}^{(2)}_{i,j} = \max(|p_1 - p_0|, |p_3 - p_2|). \end{aligned} \tag{4.22}$$

Here, $\mathcal{A}^{(1)}_{i,j}$ represents the maximum horizontal difference in the grid cell (i.e. between four neighbors), and $\mathcal{A}^{(2)}_{i,j}$ represents the maximum vertical difference.

Example 16 (Calculating $\mathcal{A}^{(1)}_{i,j}$ and $\mathcal{A}^{(2)}_{i,j}$)**.** *Consider a* 3×3 *picture. The horizontal axis uses integer coordinates* $i \in \{-1, 0, 1\}$*. The vertical axis uses integer coordinates* $j \in \{-1, 0, 1\}$*. The picture contains* 9 *pixel values, although pixel values are usually normalized between 0 and 1, in this example we use values between 0 and 255 for easier calculations. Figure 4.13 shows the pixel coordinates and their corresponding values. The bottom-left grid cell (i.e. between four neighbors) is highlighted in blue.*

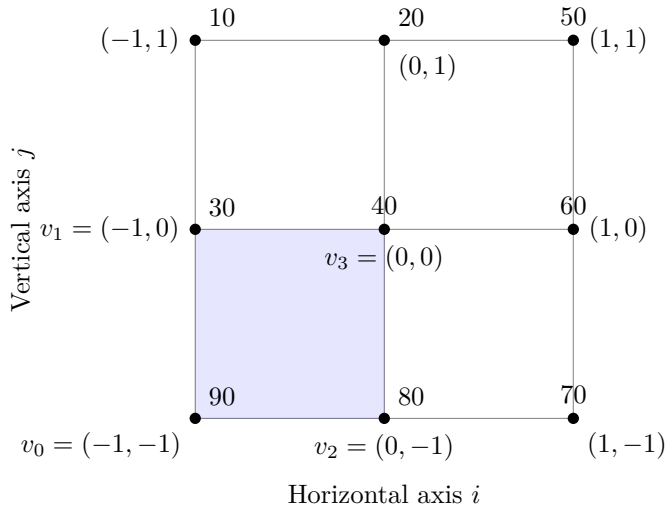


Figure 4.13: A 3×3 picture with integer coordinates and pixel values.

We calculate $\mathcal{A}^{(1)}_{i,j}$ and $\mathcal{A}^{(2)}_{i,j}$ for the highlighted bottom-left grid cell. The four nearest integer neighbors in this specific region are: $p_0 = 90$, $p_1 = 30$, $p_2 = 80$ and $p_3 = 40$. We evaluate the maximum horizontal difference $\mathcal{A}^{(1)}_{i,j}$:

$$\begin{aligned}\mathcal{A}^{(1)}_{i,j} &= \max(|p_2 - p_0|, |p_3 - p_1|) \\ &= \max(|80 - 90|, |40 - 30|) \\ &= \max(10, 10) = 10\end{aligned}$$

We also evaluate the maximum vertical difference $\mathcal{A}^{(2)}_{i,j}$ for this cell

$$\begin{aligned}\mathcal{A}^{(2)}_{i,j} &= \max(|p_1 - p_0|, |p_3 - p_2|) \\ &= \max(|30 - 90|, |40 - 80|) \\ &= \max(60, 40) = 60\end{aligned}$$

In short, the pixel intensity in this bottom-left grid cell changes by a maximum of 10 units horizontally and 60 units vertically.

To simplify the interpolation gradient formulation, we use the maximum of the gradient over the pixel domain:

$$\mathcal{A}^{(1)} = \max_{(i,j)} \mathcal{A}^{(1)}_{i,j} \qquad \mathcal{A}^{(2)} = \max_{(i,j)} \mathcal{A}^{(2)}_{i,j} \tag{4.23}$$

Note: In the 3×3 Example 16, this means calculating $\mathcal{A}^{(1)}_{i,j}$ for all 4 grid cells and taking the maximum. The same applies for $\mathcal{A}^{(2)}_{i,j}$.

By combining the bounds on the interpolation differences with the bounds on the geometric Jacobian, we obtain the bound for the lth component of $\nabla_\kappa p_{i,j}$:

$$\max_{\kappa_l} \left| \frac{\partial p_{i,j}}{\partial \kappa_l} \right| \;\leq\; \sup_{\kappa} \left| \frac{\partial i'}{\partial \kappa_l}(\kappa) \right| \cdot \mathcal{A}^{(1)} \;+\; \sup_{\kappa} \left| \frac{\partial j'}{\partial \kappa_l}(\kappa) \right| \cdot \mathcal{A}^{(2)} \tag{4.24}$$

Example 17 (Lipschitz constant for translation)**.** *Continuing from Example 15, we evaluate the bounds for the gradient of the pixel value with respect to the translation parameters t_1 and t_2.*

To establish this, we substitute the Jacobian components found in Equation (4.18) *and* (4.19) *into the general bound given by Equation* (4.24).

For the first translation parameter t_1*, the Jacobian bounds are* $\sup|\frac{\partial i'}{\partial t_1}| = 1$ *and* $\sup|\frac{\partial j'}{\partial t_1}| = 0$*. Therefore, the bound on the partial derivative is:*

$$\max_{t_1}\left|\frac{\partial p_{i,j}}{\partial t_1}\right| \leq 1 \cdot \mathcal{A}^{(1)} + 0 \cdot \mathcal{A}^{(2)} = \mathcal{A}^{(1)}$$

For the second translation parameter t_2*, the Jacobian bounds are* $\sup|\frac{\partial i'}{\partial t_2}| = 0$ *and* $\sup|\frac{\partial j'}{\partial t_2}| = 1$*. This yields:*

$$\max_{t_2}\left|\frac{\partial p_{i,j}}{\partial t_2}\right| \leq 0 \cdot \mathcal{A}^{(1)} + 1 \cdot \mathcal{A}^{(2)} = \mathcal{A}^{(2)}$$

These results show that the bounds for the translation parameters correspond to the maximum local differences of the interpolation function in the horizontal and vertical directions. Finally, we apply Equation (4.15) *to determine the Lipschitz constant for the residual function. This provides one constant per spatial dimension:*

$$\mathcal{L}_{i,j} = \left(\mathcal{A}^{(1)}, \mathcal{A}^{(2)}\right) + |\overline{a}|$$

4.2.2 STEP 2.B and STEP 2.C DeepG, PWL and Super-DeepG resolution

DeepG, PWL and Super-DeepG compute a Lipschitz constant using an analytical formulation. Super-DeepG reuses the efficient PyTorch implementation from PWL.

With the Lipschitz constant $\mathcal{L}_{i,j}$ established, we have the necessary tool to correct the unsound bound.

4.3 STEP 3: Sound linear bounds

The bound $\overline{a} \cdot \kappa + \overline{b}$ computed in Section 4.1 is unsound because it is only guaranteed to hold at the sampled discrete points, and might be violated in the continuous space between them. To ensure validity over the whole domain, we must apply a correction strategy. As established in Section 3.3.2.1 (Page 75), the function's variation between sampled points is limited by the Lipschitz constraints. Consequently, the idea is to shift the linear upper bound vertically until it sits safely above the *admissible region* (the set of all points in the space that satisfy the Lipschitz constraints, Definition 3.3) defined by these limits.

4.3.1 STEP 3A: Problem statement

The following analysis details how to mathematically quantify the exact correction term.

4.3.1.1 Correction term principle

We introduce $\overline{c}^*$ as the global minimum of the residual function over the range of perturbations $[\underline{\kappa}, \overline{\kappa}]$:

$$\overline{c}^* = \min_{\kappa \in [\underline{\kappa}, \overline{\kappa}]} \overline{r}(\kappa; \overline{a}, \overline{b}) \tag{4.25}$$

Property 4.3 (Sign of $\overline{c}^*$)**.** *As established in Proposition 3 of Step 1, there is at least one sample point* κ_m *where the residual is zero:* $\overline{r}(\kappa_m; \overline{a}, \overline{b}) = 0$*. Since* $\overline{c}^*$ *is the global minimum of the residual function over* $[\underline{\kappa}, \overline{\kappa}]$*, we have:* $\overline{c}^* \leq 0$*.*

The linear unsound bound $\overline{a} \cdot \kappa + \overline{b}$ may fail to bound the function between samples, i.e., there may exist $\kappa \in [\underline{\kappa}, \overline{\kappa}]$ such that $\overline{r}(\kappa; \overline{a}, \overline{b}) < 0$. We seek the exact correction term $\mathit{correction} \geq 0$ such that the corrected bound holds globally:

$$\forall \forall \kappa \in [\underline{\kappa}, \overline{\kappa}], \quad \overline{r}(\kappa; \overline{a}, \overline{b}) + \mathit{correction} \geq 0 \tag{4.26}$$

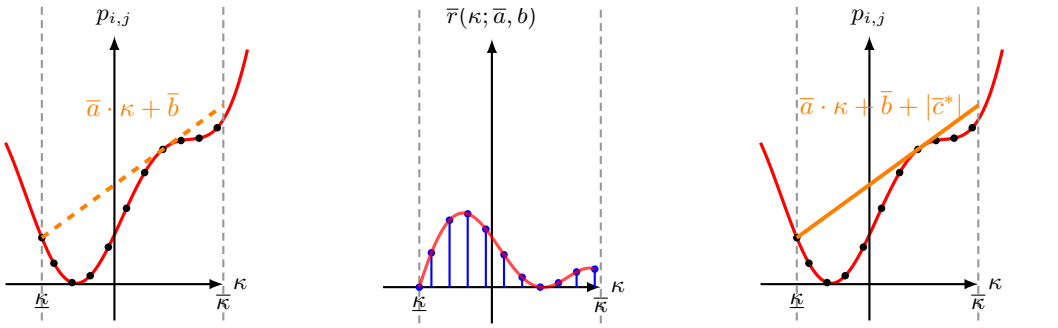
Proposition 5. *The exact correction term is the absolute value of the minimum residual:*

$$\mathit{correction} = |\overline{c}^*| \qquad \mathit{with} \quad \overline{c}^* = \min_{\kappa \in [\underline{\kappa}, \overline{\kappa}]} \overline{r}(\kappa; \overline{a}, \overline{b})$$

Proof 5. *By the definition of the minimum, for all $\kappa \in [\underline{\kappa}, \overline{\kappa}]$, we have $\overline{r}(\kappa; \overline{a}, \overline{b}) \geq \overline{c}^*$. Subtracting $\overline{c}^*$ from both sides yields $\overline{r}(\kappa; \overline{a}, \overline{b}) - \overline{c}^* \geq 0$. Therefore, $-\overline{c}^* = |\overline{c}^*|$. Substituting this gives $\overline{r}(\kappa; \overline{a}, \overline{b}) + |\overline{c}^*| \geq 0$. Thus, adding $|\overline{c}^*|$ ensures the bound is valid everywhere.*

Figure 4.14 illustrates Proposition 5 with an example where the chosen unsound bound (from the step 1) is already sound:

- In **(a)**, the initial linear bound (orange dashed line) already lies strictly above the exact function (red curve).
- In **(b)**, we plot the exact residual $\overline{r}(\kappa, \overline{a}, \overline{b})$. We observe that the curve remains above zero for the entire interval. The correction term is $\overline{c^*} = 0$.
- In **(c)**, the corrected sound bound is therefore identical to the initial unsound bound.



(a) Unsound bound and exact variation of $p_{i,j}(\kappa)$

(b) Exact variation of $\overline{r}(\kappa; \overline{a}, \overline{b})$ with $\overline{c}^* = 0$

(c) **Corrected** bound and exact variation of $p_{i,j}(\kappa)$

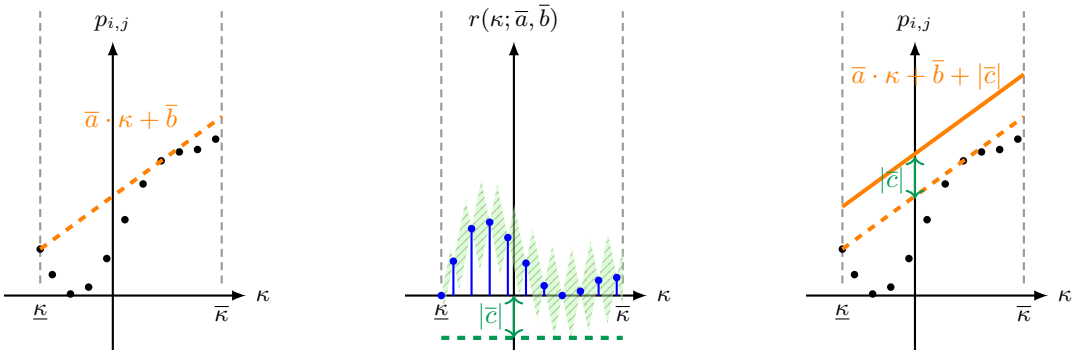
Figure 4.14: The minimal sufficient correction term is the absolute value of the minimum residual.

Hypothetically, if we knew the exact minimum of the residual, we could compute the exact correction term $|\overline{c}^*|$. In practice, we do not have access to it. Instead, we leverage the admissible region (the set of all points in the space that satisfy the Lipschitz constraints, Definition 3.3) on the residual to compute $\overline{c}$ satisfying:

$$0 \geq \overline{c}^* \geq \overline{c} \quad \Longrightarrow \quad |\overline{c}^\star| \leq |\overline{c}|$$

Note: The correction terms are negative or zero (Property 4.3); this inequality ensures that we correct at least as much as the exact correction term.

As illustrated in Figure 4.15 (b), the sound correction term $|\overline{c}|$ is identified as the global minimum of these admissible regions (in hatched green). Figure 4.15 (a) (b) details this process, showing how the correction term (b) translates into a corrected linear bound (c).



(a) Unsound bound and samples of $p_{i,j}(\kappa)$

(b) Residual $\overline{c} < 0$ and Lipschitz constraints ($\mathcal{L}_{i,j} = 5$)

(c) **Corrected** bound and samples of $p_{i,j}(\kappa)$

Figure 4.15: Determination of $\overline{c}$ using the admissible region.

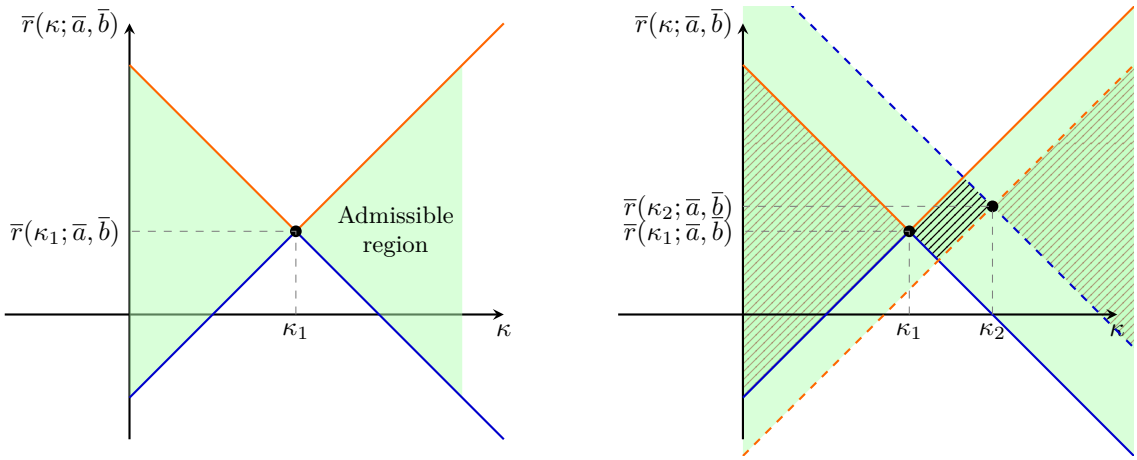
Remark 39 (From pixel value to residual function)**.** *Subsection 3.3.3 provided an intuitive visual explanation by applying Lipschitz constraints directly to the pixel value $p_{i,j}(\kappa)$ to show the necessary correction term. Here, we formalize this step mathematically. Based on Proposition 5, this correction term corresponds exactly to the minimum of the* ***residual function****. Therefore, by applying the Lipschitz constraints to the residual rather than the pixel value, we can obtain an over-approximation of the correction term needed to make the unsound bound sound.*

4.3.1.2 Quantifying the correction term $\overline{c}$

We now turn to quantifying $\overline{c}$ mathematically. We begin with the standard definition of Lipschitz continuity applied to the residual. For any two points $\kappa, \kappa_1 \in [\underline{\kappa}, \overline{\kappa}]$, the magnitude of the change in the residual is bounded by $\mathcal{L}_{i,j}$, the Lipschitz constant on $[\underline{\kappa}, \overline{\kappa}]$:

$$\|\overline{r}(\kappa; \overline{a}, \overline{b}) - \overline{r}(\kappa_1; \overline{a}, \overline{b})\| \leq \mathcal{L}_{i,j} \|\kappa - \kappa_1\|$$

We detail the calculation of the Lipschitz constant $\mathcal{L}_{i,j}$ for the residual function over $[\underline{\kappa}, \overline{\kappa}]$ in Section 4.2 (Page 100). The Lipschitz continuity visualization of Figure 3.25 (Section 3.3.2.1, Page 75) applies here to the residual function in Figure 4.16.



(a) Single reference point constraints

(b) Intersection of constraints from two points

Figure 4.16: Visualizing lipschitz constraints on the residual function. The original Figure is Figure 3.25 (Section 3.3.2.1, Page 75).

To find the minimum value, we can write:

$$-\|\overline{r}(\kappa;\overline{a},\overline{b}) - \overline{r}(\kappa_1;\overline{a},\overline{b})\| \geq -\mathcal{L}_{i,j}\|\kappa - \kappa_1\|$$

Since the residual is one-dimensional, we can use a standard property of real numbers: any number a is always greater than or equal to negative one times its absolute value ($a \geq -|a|$). Applying this to our residual difference gives:

$$\overline{r}(\kappa;\overline{a},\overline{b}) - \overline{r}(\kappa_1;\overline{a},\overline{b}) \geq -|\overline{r}(\kappa;\overline{a},\overline{b}) - \overline{r}(\kappa_1;\overline{a},\overline{b})| \geq -\mathcal{L}_{i,j}\|\kappa - \kappa_1\| \tag{4.27}$$

Since Equation (4.27) holds for all $\kappa \in [\underline{\kappa},\overline{\kappa}]$, it specifically holds for the point κ_{min} that minimizes the residual function:

$$\overline{c}^* - \overline{r}(\kappa_1;\overline{a},\overline{b}) \ \geq -\mathcal{L}_{i,j}\|\kappa_{min} - \kappa_1\| \qquad \text{with} \quad \overline{c}^* = \min_{\kappa\in[\underline{\kappa},\overline{\kappa}]} \overline{r}(\kappa;\overline{a},\overline{b}) = \overline{r}(\kappa_{min};\overline{a},\overline{b})$$

Finally, we obtain:

$$\overline{c}^* \geq \overline{r}(\kappa_1;\overline{a},\overline{b}) - \mathcal{L}_{i,j}\|\kappa_{min} - \kappa_1\| \qquad \text{with} \quad \overline{c}^* = \min_{\kappa\in[\underline{\kappa},\overline{\kappa}]} \overline{r}(\kappa;\overline{a},\overline{b}) = \overline{r}(\kappa_{min};\overline{a},\overline{b}) \tag{4.28}$$

To establish a bound that is valid for every κ in the interval $[\underline{\kappa},\overline{\kappa}]$ while maintaining a simple formulation, we apply the Lipschitz condition relative to the center of the domain, denoted as $\kappa_{mid} = \frac{\overline{\kappa}+\underline{\kappa}}{2}$. By substituting this central point into Equation (4.28), we obtain:

$$\overline{c}^* \ \geq \ \overline{r}\Big(\kappa_{mid};\overline{a},\overline{b}\Big) \ - \ \mathcal{L}_{i,j} \ \|\kappa_{min} - \kappa_{mid}\|$$

As illustrated in Figure 4.17 for the one-dimensional case, the central position of κ_{mid} ensures that the maximum deviation is bounded by the half-width of the interval. Generalizing this property to the multidimensional domain, we have: $\forall\kappa \in [\underline{\kappa},\overline{\kappa}], \frac{\|\overline{\kappa}-\underline{\kappa}\|}{2} \geq \|\kappa - \kappa_{mid}\|$. It gives $-\|\kappa - \kappa_{mid}\| \ \geq \ -\frac{\|\overline{\kappa}-\underline{\kappa}\|}{2}$. Indeed, $-\|\kappa_{min} - \kappa_{mid}\| \ \geq \ -\frac{\|\overline{\kappa}-\underline{\kappa}\|}{2}$

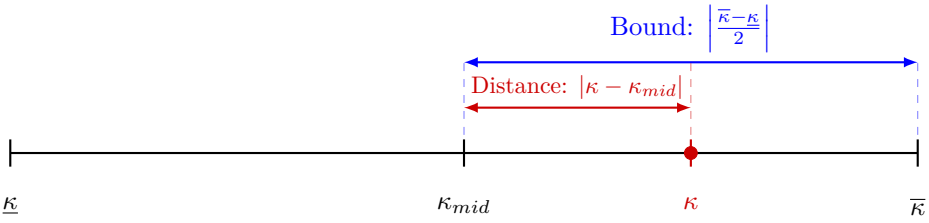


Figure 4.17: In a 1D interval $[\underline{\kappa},\overline{\kappa}]$, the distance from the midpoint κ_{mid} to any arbitrary point κ (red arrow) is less or equal to the half-width of the interval (blue arrow).

Substituting this maximum distance into our inequality yields a correction term. This value is obtained from the residual at the center and the extremities of the interval $[\underline{\kappa},\overline{\kappa}]$:

$$\overline{c}^* \ \geq \ \overline{r}\Big(\kappa_{mid};\overline{a},\overline{b}\Big) \ - \ \mathcal{L}_{i,j} \ \left\|\frac{\overline{\kappa}-\underline{\kappa}}{2}\right\| \tag{4.29}$$

Using this inequality, we define an initial correction term for the entire interval: $\overline{c}^0 = \overline{r}\Big(\kappa_{mid};\overline{a},\overline{b}\Big) \ - \ \mathcal{L}_{i,j} \ \left\|\frac{\overline{\kappa}-\underline{\kappa}}{2}\right\|$.

We now have a quantified value for the correction term that guarantees the soundness of our linear relaxation. In Equation (4.29), the Lipschitz constant $\mathcal{L}_{i,j}$ is a known input already computed in Step 2, and the interval width $\overline{\kappa}-\underline{\kappa}$ is also a fixed value. To calculate a correction term, we need to determine the residual $\overline{r}\Big(\kappa_{mid};\overline{a},\overline{b}\Big)$ at the midpoint of the interval. However, as illustrated in Figure 4.18, applying this correction term directly

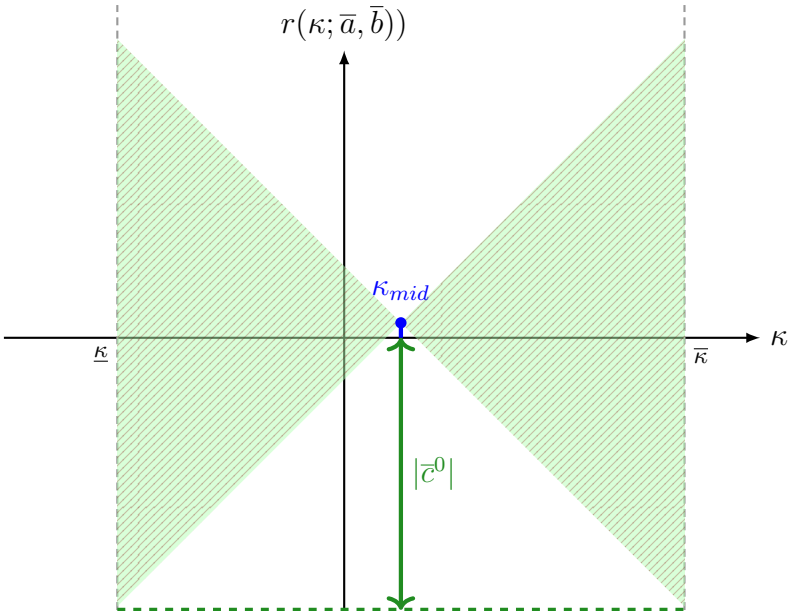


Figure 4.18: Determination of $\overline{c}$ with one sample.

over the entire range of perturbations $[\underline{\kappa}, \overline{\kappa}]$ often yields an overly conservative result (i.e., a value of $|\overline{c}^0|$ that is much larger than necessary).

We can partition $[\underline{\kappa}, \overline{\kappa}]$ into smaller sub-intervals, as formalized in the following proposition.

Proposition 6. *If we partition the interval $[\underline{\kappa}, \overline{\kappa}]$ into smaller sub-intervals such that:*

$$[\underline{\kappa}, \overline{\kappa}] = \bigcup_l \; [\underline{\kappa}_l, \overline{\kappa}_l] \tag{4.30}$$

Since the Lipschitz constant $\mathcal{L}_{i,j}$ bounds the variation over the entire domain $[\underline{\kappa}, \overline{\kappa}]$, it remains valid for any subset $[\underline{\kappa}_l, \overline{\kappa}_l]$ satisfying Equation (4.30) *Heinonen [2001]. Let $\kappa_{mid,l}$ be the midpoints of $[\underline{\kappa}_l, \overline{\kappa}_l]$, evaluating the bound (Equation* (4.29)*) across these partitions and taking the minimum provides a valid correction term:*

$$\overline{c}^* \;\geq\; \overline{c} \qquad \textit{with} \qquad \overline{c} = \min_l \left[\overline{r}\Big(\kappa_{mid,l}; \overline{a}, \overline{b}\Big) \;-\; \mathcal{L}_{i,j} \left\| \frac{\overline{\kappa}_l - \underline{\kappa}_l}{2} \right\| \right] \tag{4.31}$$

Proof 6. *Let $\overline{c}_l^*$ be the exact minimum of the residual function inside a specific sub-interval $[\underline{\kappa}_l, \overline{\kappa}_l]$. Based on Equation* (4.29)*, we calculate a valid lower bound for this specific sub-interval using its midpoint $\kappa_{mid,l}$:*

$$\overline{c}_l^* \geq \overline{r}\Big(\kappa_{mid,l}; \overline{a}, \overline{b}\Big) - \mathcal{L}_{i,j} \left\| \frac{\overline{\kappa}_l - \underline{\kappa}_l}{2} \right\|$$

The full domain $[\underline{\kappa}, \overline{\kappa}]$ is the union of all these sub-intervals, then the global minimum of the residual $\overline{c}^*$ is the smallest $\overline{c}_l^*$ found across all sub-intervals Rudin [1976]. Therefore, taking the minimum of our calculated bounds across all partitions l ensures we establish a valid correction term $\overline{c}$ for the entire domain:

$$\overline{c}^* = \min_l \overline{c}_l^* \geq \min_l \left[\overline{r}\Big(\kappa_{mid,l}; \overline{a}, \overline{b}\Big) - \mathcal{L}_{i,j} \left\| \frac{\overline{\kappa}_l - \underline{\kappa}_l}{2} \right\| \right]$$

We now compare two ways to find this correction term using Proposition 6. The difference between DeepG/PWL and Super-DeepG lies in how they select the sub-intervals defined in Equation (4.30). DeepG and PWL select the sub-intervals smartly using an iterative approach. While accurate, this sequential process cannot be easily parallelized on GPU. In contrast, Super-DeepG proposes a uniform sampling strategy.

4.3.2 STEP 3.B: DeepG and PWL resolution

To apply Proposition 6, DeepG and PWL employ an iterative *adaptive sampling* strategy, as mentioned in Subsection 3.3.2.1. The algorithm adds sample points where the admissible region is biggest, indicating range of perturbations where the correction term $|\overline{c}|$ is biggest. DeepG and PWL introduce a hyper-parameter ϵ.

This process utilizes a priority queue to target range of perturbations based on the depth of their admissible regions. We illustrate this process with a step-by-step example.

- **Initialization.** As illustrated in Figure 4.18, the process begins with the initial range of perturbations defined by a single sample point. The correction term $|\overline{c}^0|$ is computed. As this is the only range of perturbations, it is automatically assigned the highest priority in the queue (Table 4.3).

Priority	Range of perturbations	Correction term
1	$[\underline{\kappa}, \overline{\kappa}]$	$\lvert\overline{c}^0\rvert$

Table 4.3: Initialization of the priority queue.

- **First Iteration.** The algorithm removes the first sample, splits the range of perturbations in two, and inserts a new sample points at the midpoint of each sub-range of perturbations. The correction term is computed for each region, denoted as $|\overline{c}^1|$ and $|\overline{c}^2|$. In Figure 4.19, the brown arrow for $|\overline{c}^2|$ extends deeper than the orange arrow for $|\overline{c}^1|$. Consequently, this range of perturbations is assigned the highest priority (Priority 1).

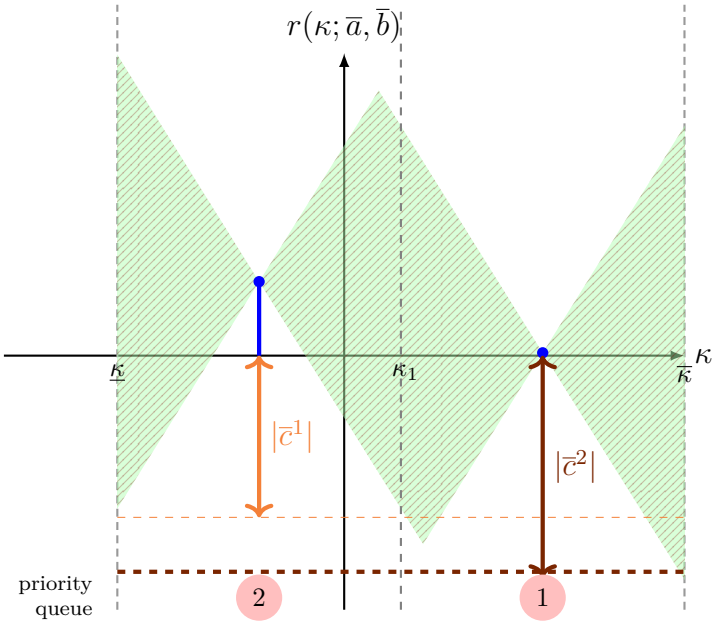


Figure 4.19: First iteration of DeepG algorithm.

Priority	Range of perturbations	Correction term
1	$[\kappa_1, \overline{\kappa}]$	$\lvert\overline{c}^2\rvert$
2	$[\underline{\kappa}, \kappa_1]$	$\lvert\overline{c}^1\rvert$

Table 4.4: Priority queue after one iteration.

- **Second Iteration.** Then, the algorithm processes the highest priority region by removing the first sample, splitting the range of perturbations in two, and inserting a new sample points at the midpoint of each sub-range of perturbations. The correction term is evaluated for each sample. In Figure 4.20, the orange arrow for $|\overline{c}^1|$ extends deeper than $|\overline{c}^3|$ and $|\overline{c}^4|$. The priority queue is updated.

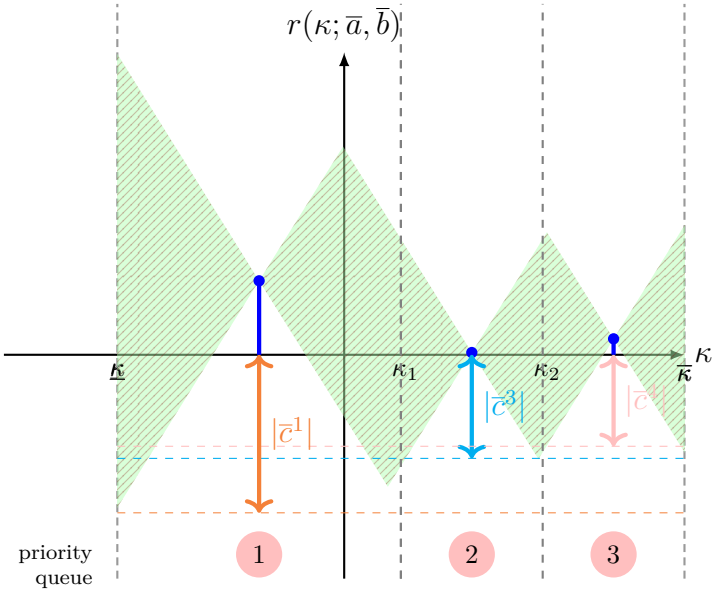


Figure 4.20: Second iteration of DeepG algorithm.

Priority	Range of perturbations	Correction term
1	$[\underline{\kappa}, \kappa_1]$	$\lvert\overline{c}^1\rvert$
2	$[\kappa_1, \kappa_2]$	$\lvert\overline{c}^3\rvert$
3	$[\kappa_2, \overline{\kappa}]$	$\lvert\overline{c}^4\rvert$

Table 4.5: Priority queue after two iterations.

DeepG and PWL repeat this cycle, inserting samples into the range of perturbations with the highest priority. It terminates when the correction term $\overline{c}$ of the highest priority provides a sufficiently tight estimate, defined with tolerance ϵ, generally very small:

$$|\overline{c}| \leq |\overline{r}\Big(\kappa_{mid_{priority1}}; \overline{a}, \overline{b}\Big) - \epsilon|$$

Figure 4.21 shows a zoomed-in view of the range of perturbations with the highest priority after many iterations, illustrating the stopping criterion using ϵ.

Based on Equation (4.29), the algorithm terminates when the sub-interval $[\kappa_l, \kappa_{l+1}]$ with the highest priority satisfies the following condition:

$$\varepsilon \geq \frac{\mathcal{L}_{i,j} \, \|\kappa_{l+1} - \kappa_l\|}{2} \tag{4.32}$$

Having explained the logic, we present the algorithm in Appendix A.1.

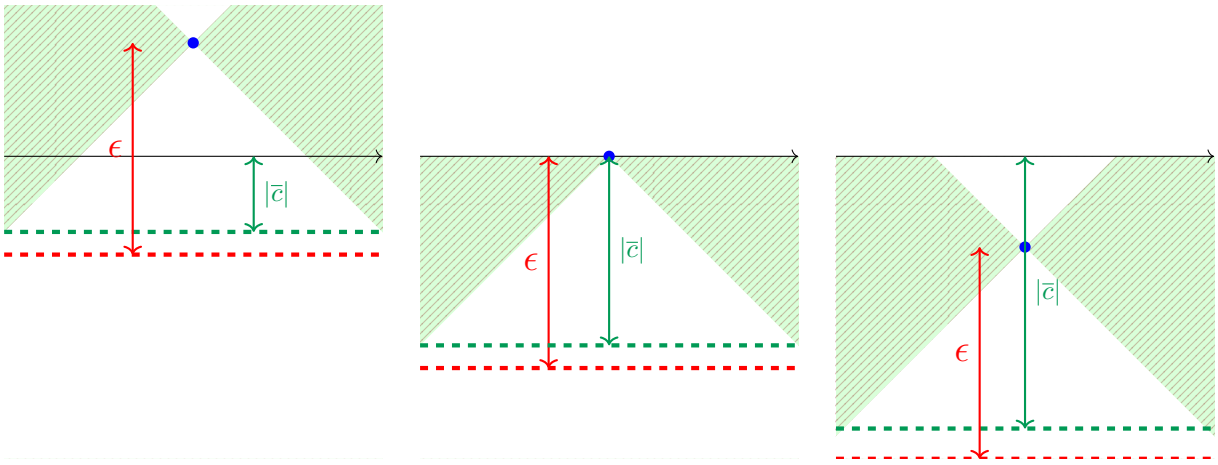


Figure 4.21: Close-up visualization of the stopping criterion for positive, zero, and negative residuals.

DeepG and PWL provide a tight correction term but rely on an iterative process that is hard to parallelize. To overcome this, Super-DeepG proposes a direct, non-iterative alternative.

4.3.3 STEP 3.C: Super-DeepG resolution - multi-dimensions

As in DeepG and PWL, the objective of this step is to compute the correction term $\overline{c}$ to ensure the upper bound is sound. Instead of using an iterative priority queue as DeepG or PWL, Super-DeepG relies on a direct, grid-based strategy. Super-DeepG introduces a hyper-parameter ϵ.

We divide the d-dimensional domain into a uniform grid, splitting each dimension into N equal segments. Figure 4.22 illustrates this grid.

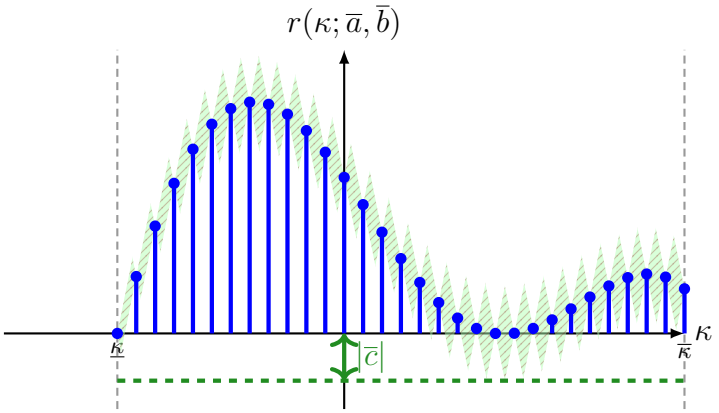


Figure 4.22: Super-DeepG approach (STEP 3) with N=32 samples.

For any given dimension, the width of each sub-interval $[\underline{\kappa}_l, \overline{\kappa}_l]$ is constant and defined as:

$$\|\overline{\kappa}_l - \underline{\kappa}_l\| = \frac{1}{N} \|\overline{\kappa} - \underline{\kappa}\| \tag{4.33}$$

Remark 40 (Number of sub-intervals)**.** *Dividing each of the d dimensions into N equal parts creates a total of N^d uniform sub-intervals across the domain.*

We apply this uniform width to the inequality from Proposition 6 (Equation (4.31)).

The uniform dimensions across all sub-intervals yield the following:

$$\begin{aligned} \overline{c} &= \min_l \left[\overline{r}\Big(\kappa_{mid,l}; \overline{a}, \overline{b}\Big) - \mathcal{L}_{i,j} \left\| \frac{\overline{\kappa}_l - \underline{\kappa}_l}{2} \right\| \right] \\ &= \min_l \left[\overline{r}\Big(\kappa_{mid,l}; \overline{a}, \overline{b}\Big) - \mathcal{L}_{i,j} \left\| \frac{\overline{\kappa} - \underline{\kappa}}{2N} \right\| \right] \end{aligned}$$

The rightmost term in the subtraction remains constant (it does not depend on the sub-interval l). Thus, we pull it outside the minimum operator:

$$\overline{c} = \min_l \left[\overline{r}\Big(\kappa_{mid,l}; \overline{a}, \overline{b}\Big) \right] - \mathcal{L}_{i,j} \left\| \frac{\overline{\kappa} - \underline{\kappa}}{2N} \right\| \tag{4.34}$$

To summarize, the computation follows these fixed steps:

- Find the center point $\kappa_{mid,l}$ for each of the N^d sub-intervals.
- Calculate the residual at every center point.
- Apply Equation (4.34) to obtain the final correction term $\overline{c}$.

We execute this exact same procedure to find the lower correction term $\underline{c}$. In the end, we secure fully sound bounds across the entire domain:

$$\forall \kappa \in [\underline{\kappa}, \overline{\kappa}], \quad \underline{a} \cdot \kappa + \underline{b} - |\underline{c}| \leq p_{i,j}(\kappa) \leq \overline{a} \cdot \kappa + \overline{b} + |\overline{c}| \tag{4.35}$$

To guarantee that Super-DeepG computes a correction term $\overline{c}$ that is at least as precise (i.e., as tight) as the one obtained by DeepG and PWL, we must properly choose the number of sub-intervals N, as explained in Property 4.4.

Property 4.4 (Equivalence to DeepG and PWL precision)**.** *DeepG and PWL rely on a priority queue. As defined in Equation* (4.32)*, their algorithm terminates when the estimated correction term for the sub-interval* $[\kappa_l, \kappa_{l+1}]$ *with the highest priority falls below the tolerance* ε*:*

$$\varepsilon \geq \frac{\mathcal{L}_{i,j} \|\kappa_{l+1} - \kappa_l\|}{2}$$

Since Super-DeepG employs a uniform grid instead of an adaptive priority queue, every sub-interval in the grid must meet this stopping condition to guarantee the same overall precision. In our uniform grid, the width of every sub-interval is fixed by Equation (4.33) *as* $\frac{\|\overline{\kappa}-\underline{\kappa}\|}{N}$*. We obtain:*

$$\varepsilon \geq \frac{\mathcal{L}_{i,j}}{2} \left(\frac{\|\overline{\kappa} - \underline{\kappa}\|}{N} \right) \implies N \geq \mathcal{L}_{i,j} \frac{\|\overline{\kappa} - \underline{\kappa}\|}{2\varepsilon} \tag{4.36}$$

By choosing an integer N *that satisfies this inequality, Super-DeepG guarantees a sound linear bound that is at least as precise as the bounds from PWL and DeepG.*

We do not detail the pseudo-code or code implementation for this step because it would just repeat the math without adding new value for the reader. In practice, we implement these equations using PyTorch. Our non-iterative method allows us to compute using parallel tensor operations on GPU. The source code for this step is available at: https://github.com/NoCohen66/SuperDeepg/layer/sound_constraint.py.

4.3.4 STEP 3: Conclusion

In this step, our primary goal was to take the unsound linear bounds and make them sound. A sound bound is guaranteed to strictly enclose the exact pixel variation $p_{i,j}(\kappa)$ for any parameter κ over the range of perturbations $[\underline{\kappa}, \overline{\kappa}]$.

To achieve this, we needed to shift the linear bounds vertically. We quantified this vertical shift, the correction term, by analyzing the minimum of the residual function. The Lipschitz constant defines an admissible region, placing a mathematical limit on how much the function can vary between our known sample points.

While DeepG and PWL find this correction term using a slow, iterative adaptive sampling process, our method relies on a direct, grid-based strategy. By dividing the multidimensional domain into a fixed number of equal sub-intervals (N^d) and checking the residual at the center of each cell, we use the Lipschitz bound to cover the maximum possible distance to the cell boundaries.

This grid-based approach guarantees the same mathematical soundness as DeepG but completely eliminates the need for sequential priority queues. It allows us to compute the correction terms $\overline{c}$ and $\underline{c}$ in a single, highly parallelizable step.

With these sound linear bounds successfully established, we now have a mathematically guaranteed over-approximation of the transformed picture ready to be evaluated by the neural network verifier.

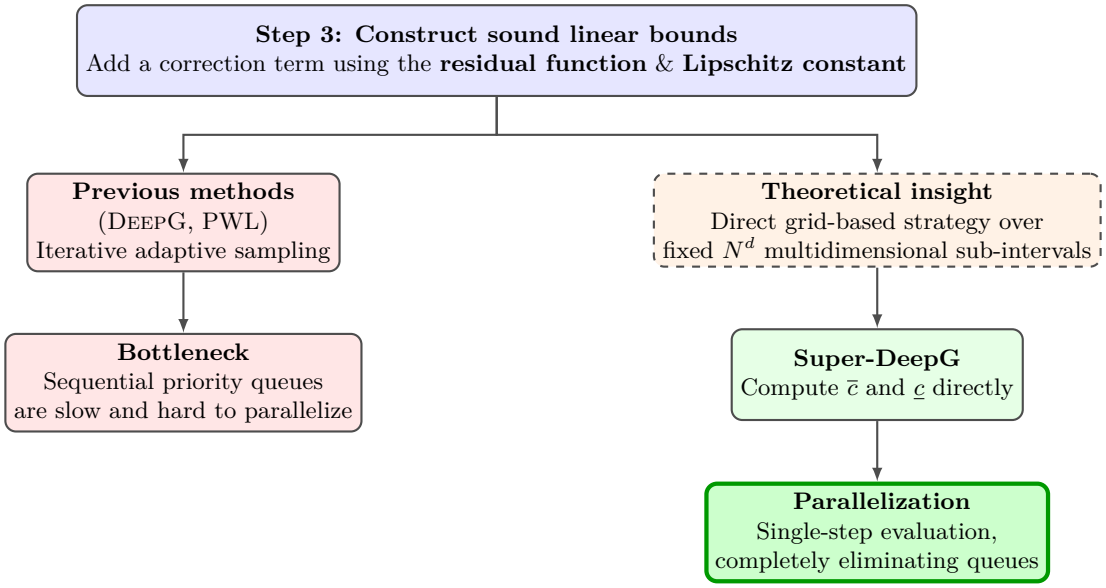


Figure 4.23: Overview of Step 3: Guaranteeing mathematical soundness for the linear bounds.

In summary, we have defined the Super-DeepG method to obtain a linear over-approximation of the set of transformed pictures. This formulation guarantees soundness while avoiding the computational cost of Gurobi solver (STEP 1) and iterative procedures (STEP 3). With the theoretical approach established, we now turn to an experimental evaluation.

4.4 Experimental evaluation

This section presents the experimental evaluation of Super-DeepG. We first describe quickly our implementation and detail the experimental setup, including the datasets and baseline methods used for comparison. We then analyze the results to demonstrate that our approach improves state-of-the-art certified accuracy and computational speed.

4.4.1 Implementation

Super-DeepG is an open-source implementation integrated within the `auto_LiRPA` Xu et al. [2020a] framework. The computation of the abstract domain $\mathcal{C}$ (described in section 4) is encapsulated in a *layer* of a special type called `CustomOp` (see Section 2.4). To verify the network's robustness (as in Equation (3.7)), users add this layer as the first layer to the neural network. This simplifies the verification process. The verification of the neural network and the geometric transformation is done with the CROWN or CROWN-IBP methods.

4.4.2 Metrics

To evaluate our method, we adopt the standard metrics used in the neural network robustness verification literature Balunovic et al. [2019]; Yang et al. [2022]. We report our results based on the following measurements:

- **Certified (%):** The percentage of test pictures guaranteed to be correctly classified under the specified transformations.
- **Time (s/picture):** The average computation time required to verify a single picture.
- **Clean accuracy (%):** The percentage of test pictures correctly classified when no geometric transformation is applied. We evaluate this metric on the same set of test pictures to establish a baseline upper bound for the certified accuracy.

During the evaluation process, we define a full range of transformation parameters for the network and the verifier to analyze. We then partition this continuous range into uniform sub-intervals. As an example, a rotation range of $[-10°, 10°]$, denoted as R(10°), with an interval size (IS) of 2° is divided into the following sub-intervals: $[-10°, -8°], [-8°, -6°], \ldots, [8°, 10°]$.

4.4.3 Experimental Setup

FGV and Super-DeepG experiments were conducted on an NVIDIA Tesla V100 PCIe (32 GB), while DeepG, lacking GPU support, was run on an Intel Xeon E5-2660 v3 @ 2.60 GHz with 62 GB of RAM and 20 cores.

4.4.3.1 Datasets

The evaluation is done on three key datasets used in prior robustness studies Balunovic et al. [2019]; Yang et al. [2022]:

- **MNIST:** Grayscale pictures of handwritten digits, 10 distinct classes. The picture size is $1 \times 28 \times 28$.
- **CIFAR10:** Color pictures categorized into 10 distinct classes. The picture size is $3 \times 32 \times 32$.

- **TinyImageNet:** Color pictures representing a subset of the larger ImageNet dataset, 200 distinct classes. The picture size is $3 \times 64 \times 64$.

4.4.3.2 Networks

We use the neural networks provided by DeepG Balunovic et al. [2019], referred to as *PGD*, and by *CGT* Yang et al. [2022], referred to as *CGT*, and by Zhang [2024] from the VNN-Comp'24 benchmark Brix et al. [2024], referred to as *VNN-Comp'24*. The PGD and CGT networks share the identical architecture for a given dataset, yet their weights differ significantly due to their distinct training strategies. CGT networks rely on a specialized geometric training dedicated to FGV. We use the VNN-Comp'24 benchmark to compare Super-DeepG and FGV on TinyImageNet without specialized geometric training. We describe the network architectures for each dataset below, representing a convolutional layer as a 4-tuple consisting of (number of filters, kernel size, stride, padding):

- **MNIST:** 2 conv layers $\{(32, 4, 2, 1), (64, 4, 2, 1)\}$ followed by 2 linear layers with $\{200, 10\}$ neurons. All layers are followed by a ReLU activation, except for the final output layer.
- **CIFAR10:** 3 conv layers $\{(32, 3, 1, 1), (32, 4, 2, 1), (64, 4, 2, 1)\}$ followed by 2 linear layers with $\{150, 10\}$ neurons. All layers are followed by a ReLU activation, except for the final output layer.
- **TinyImageNet (CGT)**[1]**:** 5 conv layers $\{(64, 3, 1, 1),\ (64, 3, 1, 1),\ (128, 3, 2, 1),\ (128, 3, 1, 1),\ (128, 3, 2, 1)\}$ followed by 2 linear layers with $\{512, 200\}$ neurons. Each conv layer is followed by a batch norm layer then a ReLU activation. The first linear layer is followed by a ReLU activation.
- **TinyImageNet (VNN-Comp'24):** 19 conv layers $\{(64, 3),\ (128, 3),\ (128, 3),\ (128, 1),\ (128, 3),\ (128, 3),\ (128, 3),\ (128, 3),\ (128, 3),\ (128, 3),\ (128, 3),\ (128, 1),\ (128, 3),\ (128, 3),\ (128, 3),\ (128, 3),\ (128, 3),\ (128, 3)\}$ followed by 2 linear layers with $\{200, 200\}$ neurons. Each conv layer is followed by a batch norm layer, with ReLU activations and addition operations interspersed to form residual blocks. The first linear layer is followed by a ReLU activation.

Depending on the specific configuration being tested, we load different weights into these architectures:

- For **MNIST**, we use 4 different weights for the DeepG configurations and 1 weight for the CGT configurations.
- For **CIFAR10**, we use 3 different weights for the DeepG configurations and 1 weight for the CGT configurations.
- For **TinyImageNet**, we use 3 different weights for the CGT configurations and 1 weight for the VNN-Comp'24 configurations.

4.4.4 Comparison methodology with state of the art

We benchmark our method against state-of-the-art verifiers: DeepG, FGV, and PWL. For PWL, although its code is available, a dependency on a MILP verifier (`VENUS` Botoeva et al. [2020]) prevented us from running it, so our comparison uses the published

[1] Pictures are resize to $3 \times 56 \times 56$.

results. We excluded Semantify-NN, as it has been proven both less precise than DeepG and slower than FGV Yang et al. [2022].

We perform the experimental validation in two stages. First, we ensure that our mathematical reformulation matches the precision of DeepG. Second, we adjust the interval size for each tool to compare runtimes at similar certification rates against FGV and PWL. Unless otherwise specified, we set the number of samples for Step 1 to $P = 10$. First, we seek to determine whether Super-DeepG matches DeepG in terms of certification rates ensuring that our reformulation performs similarly. The results are detailed in Table 4.7. Second, we want to reach competitive verification times with respect to the faster tool FGV (Tables 4.8 and 4.9). Third, we compare our tool with PWL (Table 4.10).

To guarantee a fair comparison and align with prior research, we adopt the evaluation protocols established by the authors. We evaluate the neural networks on the first 100 test pictures for the DeepG configurations as DeepG and PWL, and on the first 10000 test pictures for the CGT configurations as FGV. For VNN-Comp'24, we evaluate this neural network on the first 100 test pictures because FGV requires very small interval sizes, and therefore long computation times, to achieve good certified accuracy. Note that DeepG could not be run on the CGT neural networks for portability issues.

These results lead to five key observations.

4.4.5 Result 1: Super-DeepG is faster than DeepG, with similar (or better) accuracy.

To ensure a fair comparison, we align the hyper-parameters of Super-DeepG with those of DeepG. We match the sample count P used in their methods. Next, we determine the number of sub-intervals N based on their tolerance ε. Property 4.4 states that Super-DeepG achieves at least the same precision as DeepG if N satisfies Equation (4.36). This equation depends on the perturbation range $[\underline{\kappa}, \overline{\kappa}]$ and the Lipschitz constant. The configuration defines the perturbation range. For the Lipschitz constant, we establish an upper bound, denoted $\mathcal{L}_{\max}$. We find this by taking the maximum Lipschitz constant observed across all tested pictures in the dataset for a given configuration. Using $\mathcal{L}_{\max}$ ensures that N strictly satisfies Equation (4.36). To compare with DeepG, we use the smallest value of N that satisfies Equation (4.36). This setup guarantees that our certified accuracy will be at least as high as that of DeepG, which is confirmed by the experiments. Note that our CPU times are slower than those reported in the original paper (this is due to the use of a docker and a less performing CPU). Table 4.6 summarizes the configurations and Table 4.7 presents the corresponding results.

Config.	Dataset	Transformation	Range	Interval Size	$\mathcal{L}_{\max}$	Tolerance ε	Samples P	Grid $N_{\max}$
1A	MNIST	Rotation	$[-30°, 30°]$	1°	29.00	0.001	10	247
1B	MNIST	Rotation	$[-30°, 30°]$	6°	29.00	0.0001	1000	14500
2	CIFAR10	Rotation	$[-10°, 10°]$	1°	22.20	0.0001	1000	1887
3	MNIST	Scaling	$[-2, 2]$	4	29.41	0.006	1000	98
4	CIFAR10	Scaling	$[-1, 1]$	1	50.67	0.001	1000	254
5	MNIST	Shearing	$[-2, 2]$	4	19.12	0.006	1000	64
6	CIFAR10	Shearing	$[-2, 2]$	2	26.31	0.0001	1000	2631

Table 4.6: Parameter configurations used to guarantee Super-DeepG certified accuracy matches or exceeds DeepG. Configuration numbers in the first column link directly to the results presented in Table 4.7.

In Table 4.7, DeepG takes 210.72s per MNIST picture at $[-30°, 30°]$ to certify 98%, whereas Super-DeepG achieves 98% in 0.40s/picture, up to **526**× faster. This trend persists for CIFAR10 ($[-10°, 10°]$) with interval size of 1°, where DeepG takes 2389.70s/picture to certify 65% and Super-DeepG only 7.58s/picture with 65% certified pictures, making it approximately **315**× faster. Overall, our method consistently accelerates the computation time.

Dataset Clean accuracy (%) Transformation Config. number	**Range**	**Interval size**	**Certification method**	**Certified accuracy (%)**	**Time (s/picture)**
MNIST 98.0% Rotation n°1	$[-30°, 30°]$	1°	DeepG	98.0	210.72
			Super-DeepG	**98.0**	0.40
	$[-30°, 30°]$	6°	DeepG	86.0	48.38
			Super-DeepG	**86.0**	1.14
			Super-DeepG *	**86.0**	0.13
CIFAR10 74.0% Rotation n°2	$[-10°, 10°]$	1°	DeepG	65.0	2389.70
			Super-DeepG	**65.0**	7.58
			Super-DeepG *	**65.0**	1.90
MNIST 100.0% Scaling n°3	$[-2, 2]$	4	DeepG	98.0	4.25
			Super-DeepG	**99.0**	0.16
			Super-DeepG *	**99.0**	0.063
CIFAR10 79.0% Scaling n°4	$[-1, 1]$	1	DeepG	69.0	248.13
			Super-DeepG	**75.0**	0.83
			Super-DeepG *	**75.0**	0.20
MNIST 100.0% Shearing n°5	$[-2, 2]$	4	DeepG	98.0	4.30
			Super-DeepG	**100.0**	0.16
			Super-DeepG *	**100.0**	0.065
CIFAR10 72.0% Shearing n°6	$[-2, 2]$	2	DeepG	69.0	278.29
			Super-DeepG	**69.0**	0.71
			Super-DeepG *	**69.0**	0.18
MNIST 99.0% Translation n°7	$[-1, 1] \times [-1, 1]$	0.2	DeepG	98.0	460.93
			Super-DeepG	**98.0**	1.34

Table 4.7: Comparison of DeepG and Super-DeepG on PGD networks with equivalent parameters (see Table 4.6). *Results obtained with $P = 10$ samples.

4.4.6 Result 2: On CGT networks we match FGV's certified accuracy.

Table 4.8 compares Super-DeepG with FGV on CGT networks trained for FGV. To ensure fairness, we matched certified accuracy and compared runtimes (we adjust the interval size used by Super-DeepG so that its certified accuracy matches). On MNIST, both methods certify 94%, but FGV is significantly faster. On TinyImageNet, the runtime of Super-DeepG remains within a small factor of FGV. On CIFAR10, both reach 63% accuracy, and Super-DeepG is slightly faster, even though FGV is the fastest known method for geometric certification. This shows that while FGV performs best on networks tailored to it, Super-DeepG remains competitive in both certified accuracy and speed, while not relying on any specialized geometric training.

Dataset Clean accuracy (%) Transformation	Range	Interval size	Certified method	Certified accuracy (%)	Time (s/picture)
MNIST 99.17% Rotation	$[-30°, 30°]$	0.25°	FGV	94.2	0.0020
		2°	Super-DeepG	94.3	0.17
CIFAR10 80.47% Rotation	$[-10°, 10°]$	0.0002°	FGV	63.2	1.04
		0.0125°	Super-DeepG	63.1	0.97
TinyImageNet 27.28% Shearing	$[-2, 2]$	0.00002	FGV	18.7	0.17
		0.001	Super-DeepG	19.4	0.54
TinyImageNet 26.12% Scaling	$[-2, 2]$	0.00002	FGV	15.2	0.16
		0.001	Super-DeepG	15.9	0.48
TinyImageNet 26.01% Rotation	$[-5°, 5°]$	0.001°	FGV	13.1	0.79
		0.08°	Super-DeepG	13.0	1.28

Table 4.8: Super-DeepG performs effectively on neural networks trained for the FGV certification method (CGT network).

4.4.6.1 Result 3: Without geometric training, Super-DeepG achieves better precision and runtime than FGV

Table 4.9 shows the certification rates of FGV and Super-DeepG on PGD-trained and VNNComp'24 networks.

On networks not trained with CGT, FGV is slower and overall less accurate than Super-DeepG. For FGV, the percentage of certified pictures slightly increases as the interval size decreases, but we observe that it would not be interesting to reduce the interval size any further, since the calculation time grows rapidly and the certification percentage reaches a plateau. Meanwhile, Super-DeepG achieves significantly better certified accuracy. For example, for a rotation in $[-10°, 10°]$ on CIFAR10, FGV has a certification rate plateauing at 23% while Super-DeepG reaches 65% in 1.90s/picture.

4.4.7 Result 4: Super-DeepG is faster than PWL while maintaining competitive precision.

PWL is designed for high precision over a few wide sub-intervals, whereas our method leverages GPU parallelization to efficiently process many sub-intervals. As demonstrated in Table 4.10, when we run our method under PWL's configuration (Interval size=6, P=1000, N=14500). PWL is more precise: 91% certified accuracy versus our 86% (aligning with the 92.9% vs 87.8% reported on 98 clean pictures in their paper). However, even in this non-optimal setting, our method is already faster (1.14s/picture vs 28.3s/picture). Our method's advantage appears when we use finer splits. By splitting the domain into more sub-intervals (Interval size=1), our accuracy increases to 98%. This approach also lets us reduce other hyper-parameters (N from 14500 to 250) while maintaining 98% accuracy. Using configuration (Interval size=1, P=10, N=250), our method achieves 98% certified accuracy in 0.40s/picture, thereby outperforming the reported PWL configuration in both accuracy and runtime.

Dataset Network type Clean accuracy (%) Transformation	Range	Certified method	Interval size	Certified accuracy (%)	Time (s/picture)
CIFAR10 PGD 74.0% Rotation	$[-10°, 10°]$	FGV	2×10^{-3}	6.0	0.57
			2×10^{-4}	19.0	5.39
			2×10^{-5}	23.0	44.40
			2×10^{-6}	23.0	443
		Super-DeepG	1	**65.0**	1.90
CIFAR10 PGD 79.0% Scaling	$[-1, 1]$	FGV	5×10^{-5}	5.0	0.0436
			5×10^{-6}	19.0	0.23
			5×10^{-7}	24.0	2.02
			5×10^{-8}	24.0	19.70
			5×10^{-9}	24.0	166
		Super-DeepG	0.01	**75.0**	0.20
CIFAR10 PGD 72.0% Shearing	$[-2, 2]$	FGV	2.5×10^{-4}	3.0	0.0343
			2.5×10^{-5}	20.0	0.10
			2.5×10^{-6}	22.0	0.80
			2.5×10^{-7}	23.0	7.54
			2.5×10^{-8}	23.0	67.50
		Super-DeepG	0.02	**69.0**	0.18
MNIST PGD 98.0% Rotation	$[-30°, 30°]$	FGV	2.5×10^{-1}	0.0	0.0326
			2.5×10^{-2}	58.0	0.30
			2.5×10^{-3}	70.0	2.81
			2.5×10^{-4}	70.0	28.00
		Super-DeepG	1	**98.0**	0.40
TinyImageNet VNNComp'24 59.0% Rotation	$[-5°, 5°]$	FGV	5×10^{-4}	1.0	4.05
			4×10^{-4}	2.0	5.06
			3×10^{-4}	2.0	6.78
			2×10^{-4}	10.0	10.09
			1×10^{-4}	/	OOM
		Super-DeepG	0.02	**18.0**	3.73
TinyImageNet VNNComp'24 59.0% Scaling	$[-2, 2]$	FGV	5×10^{-6}	7.0	1.80
			1×10^{-6}	30.0	9.08
			7×10^{-7}	35.0	12.96
			6×10^{-7}	35.0	24.04
			5×10^{-7}	/	OOM
		Super-DeepG	0.0002	**36.0**	4.34
TinyImageNet VNNComp'24 59.0% Shearing	$[-2, 2]$	FGV	3×10^{-6}	38.0	2.99
			2×10^{-6}	41.0	4.48
			1×10^{-6}	47.0	8.96
			9×10^{-7}	47.0	9.95
			8×10^{-7}	47.0	11.23
		Super-DeepG	0.0003	**48.0**	3.16

Table 4.9: On networks without geometric training.

4.4.8 Result 5: Super-DeepG achieves significant geometric certification results on TinyImageNet without training specifically for geometric robustness.

While DeepG and PWL does not scale to TinyImageNet, Super-DeepG does. On neural networks not trained for FGV, Super-DeepG achieves higher certified accuracy

Dataset Clean accuracy (%) Transformation	Certified method	Config (IS / P / N)	Certified accuracy (%) (98 clean im)	Time (s/picture)
MNIST 98.0% Rotation	PWL	6 / 1000 / 14500	91.0 (92.9%=91/98)	28.30
	Super-DeepG	6 / 1000 / 14500	86.0 (87.8%=86/98)	1.14
		1 / 10 / 250	**98.0** (100.0%=98/98)	**0.40**

Table 4.10: Comparison with PWL using different configuration hyperparameters. Rotation in $[-30°, 30°]$ on MNIST.

in significantly less time than FGV. For a $[-5°, 5°]$ rotation, our method certifies 18% of pictures in 3.73s/picture, while FGV runs out of memory (OoM) attempting.

	FGV				Super-DeepG	
Interval size (°)	$4 \cdot 10^{-4}$	$3 \cdot 10^{-4}$	$2 \cdot 10^{-4}$	$1 \cdot 10^{-4}$	0.04	0.02
Certified accuracy (%)	2.0	2.0	10.0	OoM	10.0	**18.0**
Time (s/picture)	5.06	6.78	10.90	–	1.46	**3.73**

Table 4.11: On TinyImageNet VNN-Comp'24 neural network, Super-DeepG achieves higher accuracy than FGV in less time. Rotation in $[-5°, 5°]$.

4.4.9 Influence of the hyper-parameters P and interval size.

Step 1(-bis) could be considered useless, since Step 3 can correct any inaccurate bounds. Experiments still show that performing Step 1(-bis) does significantly improve the computation time. However, computing too many sample points is unnecessary as shown in Figure 4.24. We use a large interval (Interval size $= 6°$) to reveal potential variation, but observe no major improvement beyond $P = 10$. For small intervals (Interval size $= 0.5°$), certified accuracy is constant from the start.

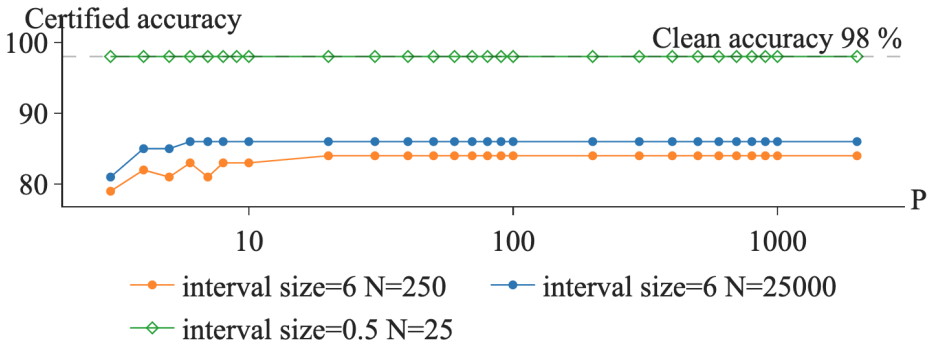


Figure 4.24: Increasing P beyond 10 brings no significant gains (log scale). Rotation in $[-30°, 30°]$ on MNIST.

4.4.10 Influence of the hyper-parameters N and interval size

Reducing the interval size improves certified accuracy significantly, reaching the clean accuracy baseline, as illustrated in Figure 4.25 with P=10 and rotations in $[-30°, 30°]$. Increasing N from 250 to 25000 yields marginal benefits while increasing computation time. The optimal trade-off is reached at an interval size of 1, where N=250 gives the optimal certified accuracy with 0.40 s/picture.

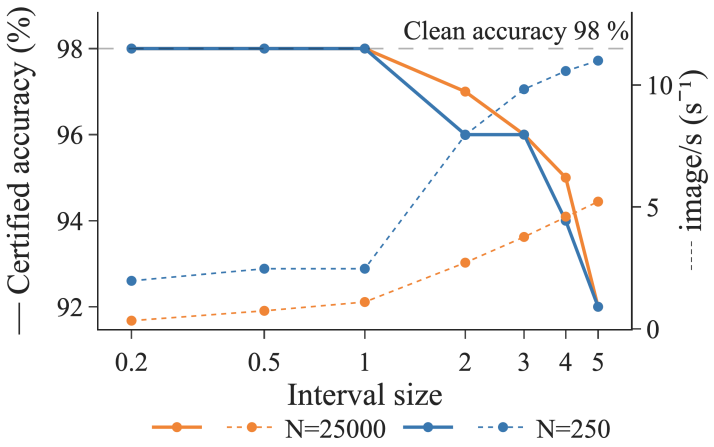


Figure 4.25: Smaller interval sizes improve certified accuracy more than increasing N (rotations in $[-30°, 30°]$ on MNIST shown), log scale.

4.5 Conclusion

We proposed Super-DeepG a method for computing linear constraints on geometric image transformations. Our work builds upon DeepG, which uses Lipschitz optimization to compute tight linear relaxations. However, both DeepG and subsequent precision-focused methods like PWL rely on computationally expensive solvers that limit their scalability. Inspired by FGV, which achieve remarkable speed via GPU-optimization and simpler interval-based abstractions, we pursued a new direction. Instead of sacrificing precision for speed, we reformulated the core optimization problems of DeepG to be analytically solvable and fully *tensorized*. This redesign eliminates computational bottlenecks and enables massive GPU parallelization. Future work will extend this framework to certify a broader set of transformations and more complex safety properties.

Chapter 5

Robustness of Object Detection to Perturbations

In the previous chapter, we showed how to verify the robustness of a neural network classifier subject to perturbations, where the output is a vector of class scores (Subsection 2.2.2). In this chapter, we extend robustness verification to object detection. This task is more complex because the output is no longer a class but a bounding box, in purple in Figure 5.1. In classification, the neural network makes a **discrete** choice to identify *what* is in the picture. Object detection, however, must determine *where* the object is by outputting **continuous** values for the bounding box coordinates. Defining and verifying robustness requires analyzing how input perturbations shift these coordinates, rather than checking if a discrete prediction changes to an incorrect label.

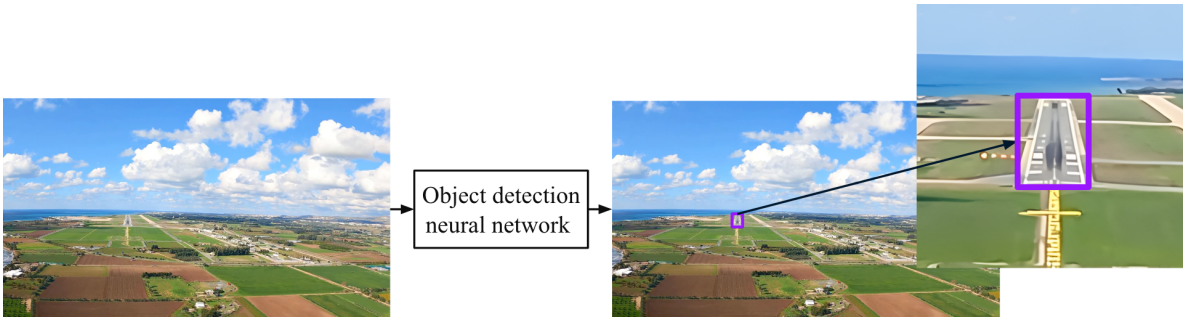


Figure 5.1: Single object detection: identifying a runway with a purple bounding box in aerial picture (Enhanced with a zoom for clarity)

Until 2023, formal methods for neural networks literature focused mainly on classification and regression. At this time, although the VNN-Comp challenge has introduced object detection, it still relies on classification-style *objectness scores* (a metric quantifying the neural network's confidence that a proposed bounding box contains an object of interest) or pixel counts in segmentation. We fill this gap by mathematically defining robustness verification for object detection. We structure this chapter into five parts:

1. We introduce the basic concepts and notations used in object detection.
2. Building on the reachability concepts from Chapter 2, Section 2.1, we apply relaxation techniques to determine the reachable set of bounding boxes when an input picture is perturbed. We define the robustness property.
3. We explain our proposed verification approach to verify the robustness property.
4. We detail the experimental setup and analyze the results, measuring how effectively the neural network continues to be correct when faced with perturbations.
5. We review existing literature and compare our methodology with recent baseline approaches in object detection robustness.

Remark 41 (Perturbation restriction)**.** *In this section, we do not consider geometric transformations as perturbations. More precisely, we limit our scope to perturbations that do not alter the position of the ground truth coordinates such as brightness, contrast, and white-noise. Geometric transformations are left for future work.*

5.1 Object detection

We first establish the basic concepts of object detection. In this section, we define the inputs and outputs of the neural network, along with the standard metric used to measure the correctness of a prediction.

Consider an object detection neural network f. As illustrated in Figure 5.1, this function takes a picture $x \in \mathbb{R}^{c \times n \times m}$ as input and outputs a bounding box $b \in \mathbb{R}^4$. We denote this bounding box as a vector of four coordinates:

$$f(x) = b = [z_0, z_1, z_2, z_3] \tag{5.1}$$

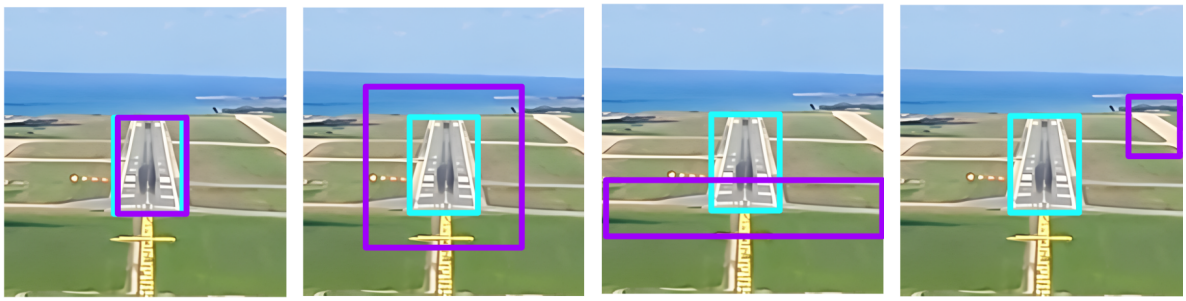
Here, (z_0, z_1) represents the 2D coordinates of the bottom-left corner, and (z_2, z_3) represents those of the top-right corner of the bounding box.

Property 5.1 (Bounding box validity)**.** *A valid bounding box requires* $z_0 \leq z_2$ *and* $z_1 \leq z_3$.

Remark 42 (Restriction to single object detection)**.** *We restrict our scope to* ***single*** *object detection:* f *returns the bounding box with the highest confidence score.*

Remark 43 (Presence of the target object)**.** *In the datasets we consider, the object of interest always appears in the picture. The neural network focuses on finding its location, so it does not need to determine whether the object is present or absent.*

In object detection, correctness is determined using a ground-truth bounding box, b_{gt}, which is accurately placed around the object. More precisely, correctness is commonly evaluated using the Intersection over Union metric that measures the match between the prediction b and b_{gt}, denoted as $\text{IoU}_{b_{gt}}(b)$. Figure 5.2 illustrates the quality of this match between b (purple) and b_{gt} (turquoise): $\text{IoU}_{b_{gt}}(b) = 1$ represents a perfect match, while $\text{IoU}_{b_{gt}}(b) = 0$ signifies that the boxes do not overlap. Definition 5.2 defines the $\text{IoU}_{b_{gt}}(b)$ and Definition 5.1 formalizes how $\text{IoU}_{b_{gt}}(b)$ defines the correctness of b.



$\text{IoU}_{b_{gt}}(b) = 0.98$ $\text{IoU}_{b_{gt}}(b) = 0.24$ $\text{IoU}_{b_{gt}}(b) = 0.09$ $\text{IoU}_{b_{gt}}(b) = 0.00$

Figure 5.2: Comparison of different Intersection over Union (IoU) values.

Definition 5.1 (Correctness of a bounding box)**.** *Let* τ *be a safety threshold. A bounding box prediction* b *is considered correct with regard to a ground truth bounding box* b_{gt} *when the following condition is satisfied:*

$$IoU_{b_{gt}}(b) \geq \tau \tag{5.2}$$

Remark 44 (Safety threshold)**.** *In object detection benchmarks, the safety threshold is commonly set to* $\tau = 0.5$*. In Figure 5.2, only the example on the left satisfies this condition.*

$\text{IoU}_{b_{gt}}$, also known as Jaccard Index Rezatofighi et al. [2019], is a metric that quantifies box overlap by calculating the ratio of their intersection area to their union area.

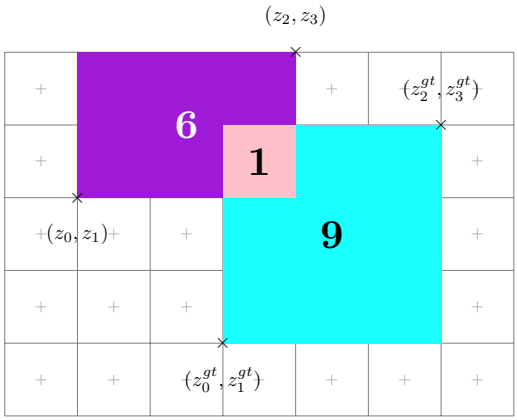
Definition 5.2 (Intersection over Union – $\text{IoU}_{b_{gt}}$)**.** *Let a bounding box* $b = [z_0, z_1, z_2, z_3] \in \mathbb{R}^4$ *and its area* $a(b)$ *be defined by* $a : \mathbb{R}^4 \longmapsto \mathbb{R}_+$ *where* $a(b) = (z_2 - z_0) \times (z_3 - z_1)$*. Let a ground truth bounding box* $b_{gt} = [z_0^{gt}, z_1^{gt}, z_2^{gt}, z_3^{gt}]$ *and its area* $a(b_{gt})$ *be similarly defined. Let the intersection of* b *with* b_{gt}*,* $i_{b_{gt}}(b)$*, be defined by the function* $i : \mathbb{R}^4 \longmapsto \mathbb{R}^4$*,*

$$i_{b_{gt}}(b) = \Big(\max(z_0, z_0^{gt}),\ \ \max(z_1, z_1^{gt}),\ \ \min(z_2, z_2^{gt}),\ \ \min(z_3, z_3^{gt}) \Big) \tag{5.3}$$

and its area $a(i_{b_{gt}}(b))$*. The IoU is a function* $\mathbb{R}^4 \longmapsto [0, 1]$ *such that:*

$$IoU_{b_{gt}}(b) = \frac{a(i_{b_{gt}}(b))}{a(b_{gt}) + a(b) - a(i_{b_{gt}}(b))} \tag{5.4}$$

Figure 5.3 provides an example.



We consider a tiny picture of width $n = 7$ and height $m = 5$. The bounding boxes are filled to visualize their areas. Let b_{gt} be the turquoise and b the purple bounding boxes, such that:

- Intersection (pink): $b_{gt} \cap b = i_{b_{gt}}(b)$.
- Areas:
  - $a(b) = 6$ (purple)
  - $a(b_{gt}) = 9$ (turquoise)
  - $a(i_{b_{gt}}(b)) = 1$ (pink)
- $IoU_{b_{gt}}(b) = \frac{1}{9+6-1} = \frac{1}{14} \simeq 0.07$

Figure 5.3: Quantitative example of $\text{IoU}_{b_{gt}}(b)$ calculation with predicted bounding box $b = [-2, 1, 0, 2]$ and ground truth $b_{gt} = [0, -1, 2, 1]$. The grid is indexed by integer coordinates centered at $(0, 0)$ in the center of the picture.

Property 5.2 (Differentiability of $\text{IoU}_{b_{gt}}(b)$)**.** *The function* $IoU_{b_{gt}}(b)$ *has the following differentiability properties:*

- *The* min *and* max *functions in Equation* (5.3) *are not differentiable when* $z_k = z_k^{gt}$*, which makes* $IoU_{b_{gt}}(b)$ *non-differentiable at these points for any* $k \in \{0, \ldots, 3\}$*.*
- *It is infinitely differentiable everywhere else.*

Thus, $IoU_{b_{gt}}(b)$ *is piecewise differentiable.*

Now that we have established how to measure the correctness of a bounding box using the Intersection over Union metric, we can formally define the verification problem.

5.2 Problem statement

We consider an object detection neural network f and an input picture x. Our goal is to verify the robustness of the neural network when the picture faces perturbations, such as white-noise or changes in brightness.

5.2.1 Relaxation of the object detection neural network

As explained in the chapter Background, the perturbations around x are represented by an input domain $\mathcal{C}$. We characterize the set of reachable bounding boxes $f(\mathcal{C}) = \{b \in \mathbb{R}^4 \mid b = f(x),\ x \in \mathcal{C}\}$. Since computing the exact image $f(\mathcal{C})$ is intractable, we use relaxations like CROWN or IBP to find an over-approximation $f^a(\mathcal{C})$ (Section 2.3.3). The relaxation of the network f over the input domain $\mathcal{C}$, denoted $f^a(\mathcal{C})$, results in interval bounds for each of the four coordinates:

$$f^a(\mathcal{C}) = \left\{ b \in \mathbb{R}^4 \;\middle|\; \underline{z_k} \leq z_k \leq \overline{z_k}, \quad \forall k \in 0, \ldots, 3 \right\} \tag{5.5}$$

The top plot of Figure 5.4 shows the inference result for the reference input $x^\star$. The predicted bounding box is defined with respect to the coordinate axes (i, j) by:

- its bottom-left corner (z_0, z_1), the red point, and
- its top-right corner (z_2, z_3), the blue point.

The bottom plot of Figure 5.4 visualizes $f^a(\mathcal{C})$ (Equation (5.5)). Any perturbed picture $x \in \mathcal{C}$ results in a bounding box that is guaranteed to vary within specific limits:

- its bottom-left corner remains inside $[\underline{z_0}, \overline{z_0}] \times [\underline{z_1}, \overline{z_1}]$, the red rectangle, and
- its top-right corner remains inside $[\underline{z_2}, \overline{z_2}] \times [\underline{z_3}, \overline{z_3}]$, the blue rectangle.

The relaxation results in an infinite number of potential bounding boxes; the bottom Figure 5.4 only shows one. Figure 5.5 illustrates other examples of bounding boxes included in $f^a(\mathcal{C})$.

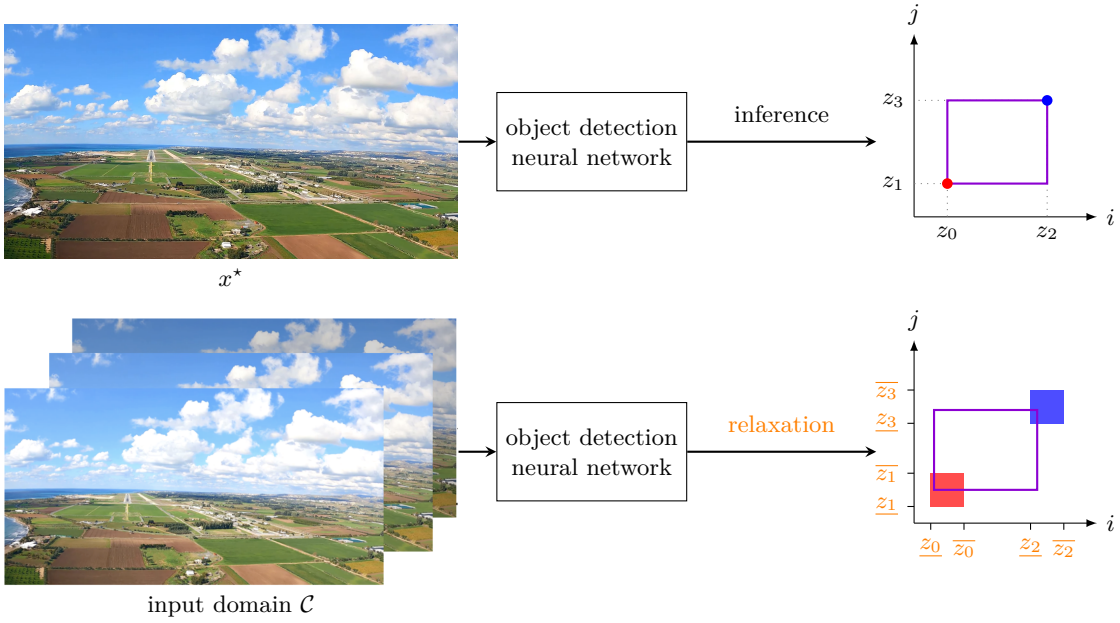


Figure 5.4: Illustration of the relaxation of an object detection neural network. Top: Inference results in a single bounding box. Bottom: The relaxation results in an infinity of potential bounding boxes, whose bottom-left corner is within the red region and whose top-right corner within the blue region.

Property 5.3 (Over-approximation validity)**.** *To guarantee that every bounding box $b \in f^a(\mathcal{C})$ satisfies Property 5.1, the bounds of the over-approximation must not allow for inverted corners. Therefore, it must hold that $\overline{z}_0 \leq \underline{z}_2$ and $\overline{z}_1 \leq \underline{z}_3$.*

While we have successfully computed intervals for the output, a key question remains: **how do we guarantee robustness of f?**

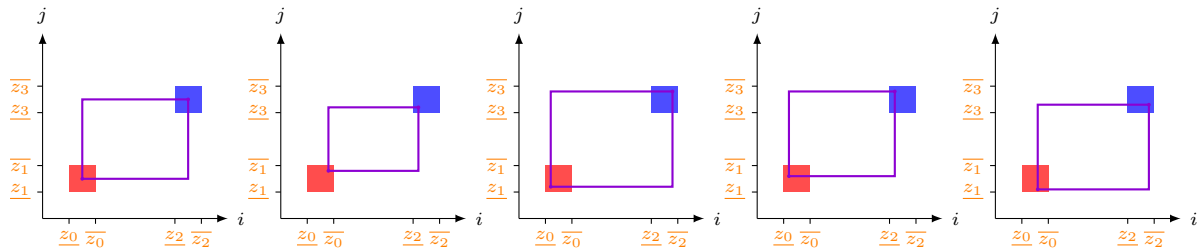


Figure 5.5: Five examples of bounding boxes included in the set $f^a(\mathcal{C})$.

5.2.2 Robustness verification

To prove that f is robust, we must prove that every possible bounding box in the over-approximation $f^a(\mathcal{C})$ is correct.

Remark 45 (b_{gt} is fixed)**.** *In this section, b_{gt} is fixed. As explained in Remark 41, we consider perturbations that do not alter the position of the ground truth coordinates.*

Checking that five specific bounding boxes are correct is straighforward using Definition 5.1. Figure 5.6 illustrates this for a safety threshold $\tau = 0.5$. However, the set $f^a(\mathcal{C})$ contains an infinite number of possible bounding boxes.

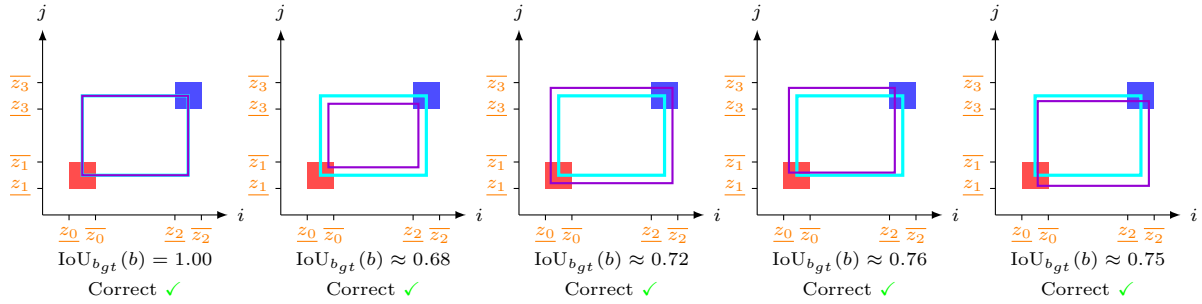


Figure 5.6: Correctness for five examples included in $f^a(\mathcal{C})$. The ground truth b_{gt} is fixed (turquoise).

Following the reachability viewpoint from Section 2.2.2, we define a specification set Y_{gt} that contains all correct bounding boxes. We show that the neural network is robust if the over-approximation $f^a(\mathcal{C})$ is inside this set ($f^a(\mathcal{C}) \subseteq Y_{gt}$). Given a ground truth b_{gt} and the correctness definition in Equation (5.2), the specification set Y_{gt} is:

$$Y_{gt} = \left\{ b \in \mathbb{R}^4 \;\middle|\; \text{IoU}_{b_{gt}}(b) \geq \tau \right\}$$

To verify $f^a(\mathcal{C}) \subseteq Y_{gt}$, we must ensure that even the worst bounding box prediction (the least accurate bounding box) in $f^a(\mathcal{C})$ is considered correct:

$$\left(\min_{b \in f^a(\mathcal{C})} \text{IoU}_{b_{gt}}(b) \right) \geq \tau \tag{5.6}$$

Finally, the problem statement for verifying the robustness of the neural network f requires checking that the minimum $\text{IoU}_{b_{gt}}$ over all possible perturbed pictures $x \in \mathcal{C}$ meets the safety threshold:

$$\left(\min_{b \in f^a(\mathcal{C})} \text{IoU}_{b_{gt}}(b) \right) \geq \tau \quad \Rightarrow \quad \left(\min_{x \in \mathcal{C}} \text{IoU}_{b_{gt}}(f(x)) \right) \geq \tau \tag{5.7}$$

Computing this guaranteed minimum $\text{IoU}_{b_{gt}}$ value is a difficult problem. The function depends on the variation of four coordinates z_k within the interval bounds $[\underline{z_k}, \overline{z_k}]$ for all $k \in \{0, \ldots, 3\}$ (following Equation (5.5)). Figure 5.7 illustrates $\text{IoU}_{b_{gt}}(b)$ in 3D. We observe how the predicted bounding box varies around a known ground truth bounding box: $[z_0^{gt}, z_1^{gt}, z_2^{gt}, z_3^{gt}] = [0, 0, 1, 1]$. We keep the top-right corner fixed at $z_2 = 1$ and $z_3 = 1$. To see how the IoU behaves, we allow the bottom-left coordinates, z_0 and z_1, to move within a margin of ± 1 around the ground truth coordinates. Specifically, z_0 varies within the interval $[z_0^{gt} - 1, z_0^{gt} + 1]$ and z_1 varies within $[z_1^{gt} - 1, z_1^{gt} + 1]$.

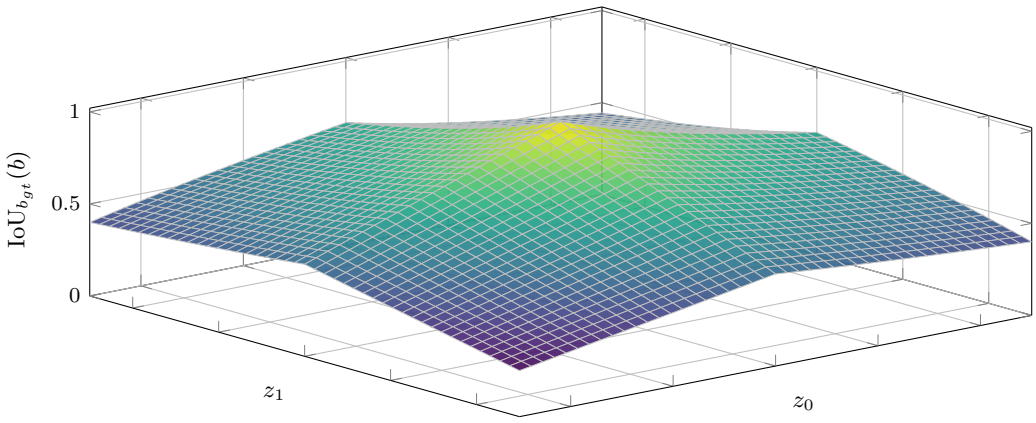


Figure 5.7: 3D visualization of $\text{IoU}_{b_{gt}}(b)$ (z_2, z_3 fixed).

The surface of this graph is more complex than it seems and is not convex. Finding the minimum value is not as straightforward as in classification. This verification requires an additional post-processing analysis. We decompose the process into two steps, as illustrated in Figure 5.8:

- **Step 1:** Compute $f^a(\mathcal{C})$. We use verification tools (e.g. `auto-LiRPA`) to find these bounds.
- **Step 2:** Find $\min_{b \in f^a(\mathcal{C})} \text{IoU}_{b_{gt}}(b)$ to verify Equation (5.6).

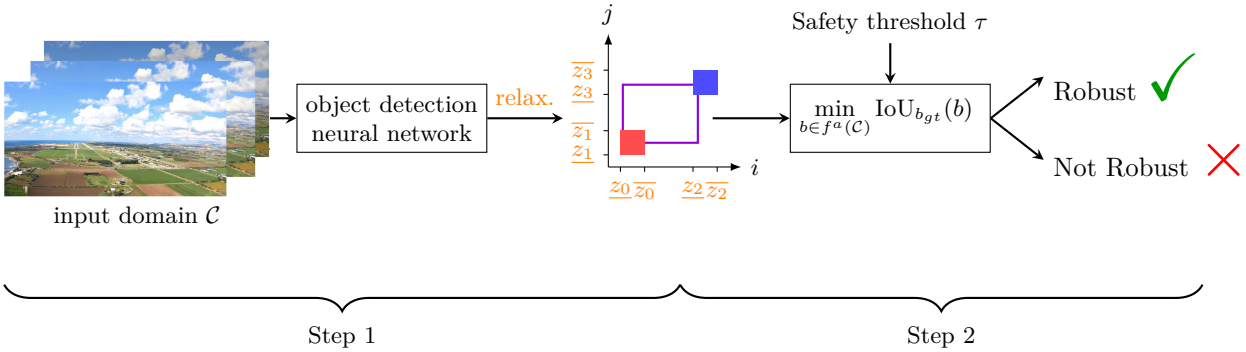


Figure 5.8: Illustration of the relaxation and verification of an object detection neural network.

Remark 46 (Concretization of coordinates)**.** *Although the verification tools (Step 1) can provide linear bounds for coordinates, we use intervals coordinates $[\underline{z_k}, \overline{z_k}]$ (Equation 5.5) for Step 2.*

Computing the guaranteed minimum $\text{IoU}_{b_{gt}}$ value in Step 2 was an open problem in formal verification. To address this, our main contributions are as follows:

1. **Object detection verification:** Since the behavior of the $\text{IoU}_{b_{gt}}(b)$ metric under formal verification bounds has not been explored in the literature, our primary contribution here is an analysis of its properties. Then, we introduce our a novel strategy to determine $\min_{b \in f^a(\mathcal{C})} \text{IoU}_{b_{gt}}(b)$ in order to verify Equation (5.6).

2. **Real-world use case and perturbations:** We demonstrate the effectiveness of our approach on a real-world industrial dataset and a standard academic benchmark. For the experiments, we evaluated neural network robustness against real-world perturbations, specifically brightness, contrast and white-noise. While the mathematical encoding of these perturbations was originally proposed by Elboher et al. [2024a] specifically for classification task using the Marabou solver, we adapt and integrate these formulations into `auto-LiRPA` for object detection. We achieve this using wrapper layers (as introduced in Example 10). Importantly, this implementation can be reused for other tasks (i.e classification or regression). We provide an open-source implementation[1].

3. **Related work comparison:** At the time of our 2023 submission, which we published as a preprint on arXiv, Raviv et al. [2024] also expanded on this topic and compared their approach with ours. Since our approach achieved better results, later work by Brückner et al. [2026] built upon it. We outline their method and detail how it differs from our own.

Let us now focus on the first contribution. To properly verify the robustness property, we must first understand how the metric behaves when the predicted bounding box varies within the interval bounds.

5.3 Verification of object detection neural networks

Calculating the guaranteed minimum value requires us to analyze the mathematical properties of $\text{IoU}_{b_{gt}}(b)$.

5.3.1 Strategies for robustness verification

Calculating $\min_{b \in f^a(\mathcal{C})} \text{IoU}_{b_{gt}}(b)$ constitutes a constrained optimization problem. To find the minimum value of a function, we generally observe its properties (linearity, convexity, etc.) and then apply the corresponding optimization method. As shown in Figure 5.9, we consider four standard properties used in optimization to find guaranteed minimums Boyd and Vandenberghe [2004]:

[1] https://github.com/NoCohen66/Verification4ObjectDetection

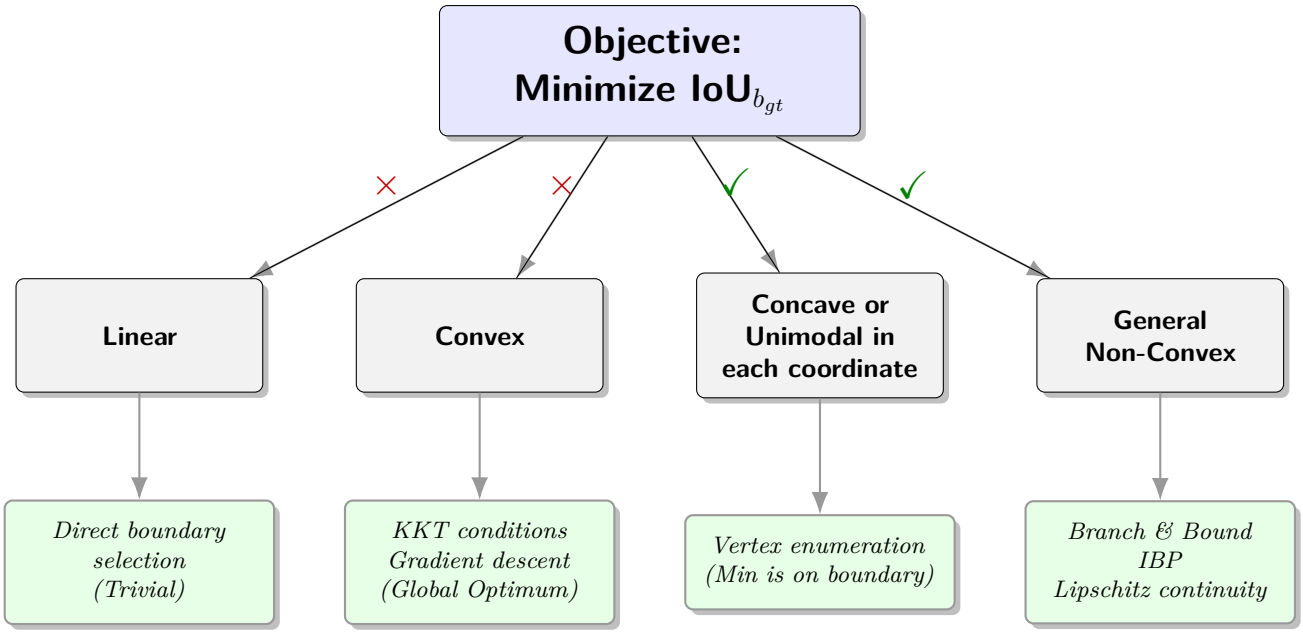


Figure 5.9: Decision tree for selecting a **sound** minimization strategy based on function properties.

- **Linear function:** If the function is linear, the problem would be trivial. Since variables are independent, the global minimum is found by selecting the lower or upper bound for each coordinate based on the sign of its coefficient. However, as seen in Equation (5.4), $\text{IoU}_{b_{gt}}(b)$ is non-linear. It is a rational function (a ratio of areas) and includes $\min, \max$ operations in the intersection term.

- **Convex function:** For a convex function, we can use analytical solutions (e.g. KKT conditions) or gradient-based methods to find the global minimum. However, **the $\text{IoU}_{b_{gt}}(b)$ is not convex**, as we demonstrate in the next section.

- **Concave or Unimodal function:** For these functions, the minimum is always located at the vertices (the corners) of the input intervals. Evaluating the function at these points is sufficient. We show in the next section that **the $\text{IoU}_{b_{gt}}(b)$ is unimodal in each coordinate**.

- **General non-convex function:** If the function is neither convex nor concave, or if its properties are too difficult to prove, we treat it as a general non-convex problem. Since Equation (5.4) relies on elementary operations (addition, subtraction, division, max), we can apply IBP. As explained in Subsection 2.3.5, this technique propagates the interval bounds from $f^a(\mathcal{C})$ directly through the formula to obtain a lower bound.

Based on this analysis, we propose an original verification method: an exact evaluation based on a unimodality proof. We operate under Hypothesis 5.1.

Hypothesis 5.1. *If a predicted bounding box b and the ground truth b_{gt} do not overlap, then $IoU_{b_{gt}}(b) = 0$ (as shown in Figure 5.2), meaning the robustness property (Equation (5.6)) is trivially not verified. Therefore, the following section assumes that all predicted bounding boxes within the perturbation domain $f^a(\mathcal{C})$ overlaps with the ground truth bounding box.*

5.3.1.1 The $\text{IoU}_{b_{gt}}(b)$ is not convex nor concave

We prove $\text{IoU}_{b_{gt}}(b)$ is neither convex nor concave using the second-order condition. Convex optimization tools cannot be applied to find $\min_{b \in f^a(\mathcal{C})} \text{IoU}_{b_{gt}}(b)$ (Equation (5.6)).

Property 5.4 (Hessian characterization of convexity and concavity)**.** *For a twice differentiable function, convexity requires the Hessian matrix H to be positive semidefinite ($v^\top Hv \geq 0$ for all vectors v) everywhere. Conversely, concavity requires the Hessian to be negative semidefinite ($v^\top Hv \leq 0$ for all vectors v) everywhere.*

To apply Property 5.4, we restrict our analysis to the open regions where $\text{IoU}_{b_{gt}}(b)$ is infinitely differentiable (i.e., $z_k \neq z_k^{gt}$ for all k) as stated in Property 5.2. In these regions, the function is twice differentiable, allowing us to define the partial derivatives with respect to the coordinates of $b = [z_0, z_1, z_2, z_3]$. For each $k \in \{0, \ldots, 3\}$, we denote the partial derivative by $\frac{\partial \text{IoU}_{b_{gt}}(b)}{\partial z_k}$. The gradient is then defined locally as:

$$\nabla \text{IoU}_{b_{gt}}(b) = \Big[\frac{\partial \text{IoU}_{b_{gt}}(b)}{\partial z_0}, \ \frac{\partial \text{IoU}_{b_{gt}}(b)}{\partial z_1}, \ \frac{\partial \text{IoU}_{b_{gt}}(b)}{\partial z_2}, \ \frac{\partial \text{IoU}_{b_{gt}}(b)}{\partial z_3}\Big].$$

The Hessian matrix of $\text{IoU}_{b_{gt}}(b)$ is the 4×4 matrix whose (k, ℓ) entry is $\frac{\partial^2 \text{IoU}_{b_{gt}}(b)}{\partial z_k \, \partial z_\ell}$:

$$\nabla^2 \text{IoU}_{b_{gt}}(b) = \begin{pmatrix} \frac{\partial^2 \text{IoU}_{b_{gt}}(b)}{\partial z_0^2} & \frac{\partial^2 \text{IoU}_{b_{gt}}(b)}{\partial z_0 \partial z_1} & \frac{\partial^2 \text{IoU}_{b_{gt}}(b)}{\partial z_0 \partial z_2} & \frac{\partial^2 \text{IoU}_{b_{gt}}(b)}{\partial z_0 \partial z_3} \\ \frac{\partial^2 \text{IoU}_{b_{gt}}(b)}{\partial z_1 \partial z_0} & \frac{\partial^2 \text{IoU}_{b_{gt}}(b)}{\partial z_1^2} & \frac{\partial^2 \text{IoU}_{b_{gt}}(b)}{\partial z_1 \partial z_2} & \frac{\partial^2 \text{IoU}_{b_{gt}}(b)}{\partial z_1 \partial z_3} \\ \frac{\partial^2 \text{IoU}_{b_{gt}}(b)}{\partial z_2 \partial z_0} & \frac{\partial^2 \text{IoU}_{b_{gt}}(b)}{\partial z_2 \partial z_1} & \frac{\partial^2 \text{IoU}_{b_{gt}}(b)}{\partial z_2^2} & \frac{\partial^2 \text{IoU}_{b_{gt}}(b)}{\partial z_2 \partial z_3} \\ \frac{\partial^2 \text{IoU}_{b_{gt}}(b)}{\partial z_3 \partial z_0} & \frac{\partial^2 \text{IoU}_{b_{gt}}(b)}{\partial z_3 \partial z_1} & \frac{\partial^2 \text{IoU}_{b_{gt}}(b)}{\partial z_3 \partial z_2} & \frac{\partial^2 \text{IoU}_{b_{gt}}(b)}{\partial z_3^2} \end{pmatrix}.$$

Proposition 7. *The function $IoU_{b_{gt}}(b)$ is neither convex nor concave in $b \in \mathbb{R}^4$.*

Proof 7 (Proof of Proposition 7 by counter-example)**.** *Fix $b_{gt} = [z_0^{gt}, z_1^{gt}, z_2^{gt}, z_3^{gt}] = [0, 0, 2, 2]$, so $a(b_{gt}) = 4$. Fix $(z_2, z_3) = (1, \frac{3}{2})$ and consider $b = [z_0, z_1, 1, \frac{3}{2}]$ with $0 < z_0 < 1$ and $0 < z_1 < \frac{3}{2}$. Then b is strictly inside b_{gt}, so Equation (5.3) gives $i_{b_{gt}}(b) = b$, and Equation (5.4) simplifies to*

$$IoU_{b_{gt}}(b) = \frac{a(b)}{a(b_{gt})} = \frac{(1 - z_0)\,(\frac{3}{2} - z_1)}{4}.$$

This function is twice differentiable in (z_0, z_1). Its Hessian matrix with respect to (z_0, z_1) is

$$\begin{pmatrix} \frac{\partial^2 IoU_{b_{gt}}(b)}{\partial z_0^2} & \frac{\partial^2 IoU_{b_{gt}}(b)}{\partial z_0 \partial z_1} \\ \frac{\partial^2 IoU_{b_{gt}}(b)}{\partial z_1 \partial z_0} & \frac{\partial^2 IoU_{b_{gt}}(b)}{\partial z_1^2} \end{pmatrix} = \begin{pmatrix} 0 & \frac{1}{4} \\ \frac{1}{4} & 0 \end{pmatrix}.$$

For the direction $(1, 1)$, we get

$$(1, 1) \begin{pmatrix} 0 & \frac{1}{4} \\ \frac{1}{4} & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{1}{2} > 0,$$

and for the direction $(1, -1)$, we get

$$(1, -1) \begin{pmatrix} 0 & \frac{1}{4} \\ \frac{1}{4} & 0 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = -\frac{1}{2} < 0.$$

Hence this Hessian matrix is neither positive semidefinite nor negative semidefinite. Therefore $IoU_{b_{gt}}(b)$ is neither convex nor concave in b.

Having excluded convexity and concavity, we now move to the next branch of our strategy (Figure 5.9) by investigating the function's behavior along each individual axis.

5.3.1.2 The $\text{IoU}_{b_{gt}}(b)$ is unimodal in each coordinate

To the best of our knowledge, we are the first to mathematically demonstrate that the $\text{IoU}_{b_{gt}}(b)$ function is unimodal in each coordinate within the context of interval bounds (Equation (5.5). In this section, we detail this proof and show how it simplifies the verification problem.

Proposition 8 (Unimodality per coordinate)**.** *The function $IoU_{b_{gt}}(b)$ is unimodal with respect to each coordinate independently. For any index $k \in \{0, \ldots, 3\}$, fixing the other three coordinates, denoted $z_{\setminus k}$, yields a univariate function $IoU_{b_{gt}}(z_k; z_{\setminus k})$ that has exactly one local maximum at a point z_k^*. This function is monotonically increasing for $z_k \leq z_k^*$ and monotonically decreasing for $z_k \geq z_k^*$*

We visualize this behavior in Figure 5.10, which illustrates how the function reaches its maximum when the coordinate aligns with the ground truth ($z_k = z_k^{gt}$), while all other coordinates remain fixed.

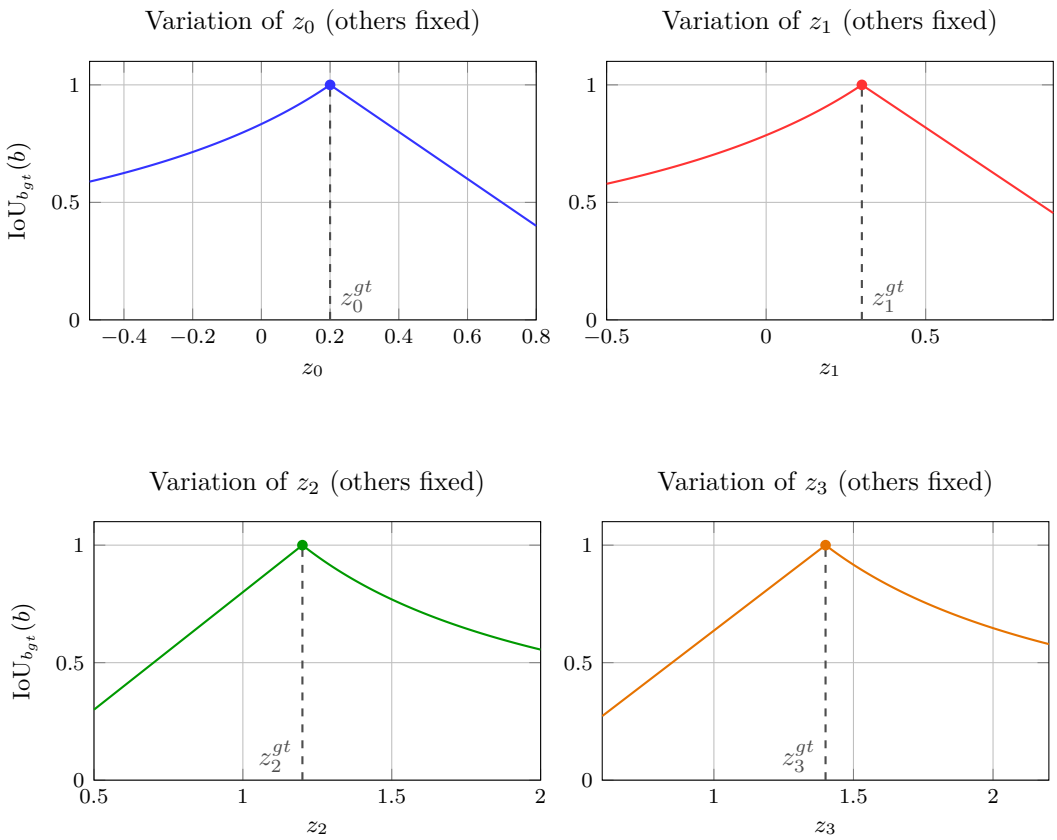


Figure 5.10: Univariate analysis of $\text{IoU}_{b_{gt}}(b)$. Each subplot shows the function's variation when one coordinate is modified and the other three are fixed to their ground truth values.

To establish Proposition 8, we prove that $\text{IoU}_{b_{gt}}(b)$ is unimodal by analyzing the signs of its partial derivatives. For any given coordinate z_i and its corresponding ground truth value z_i^{gt}, we consider two cases:

- First, we show that when $z_i < z_i^{gt}$, the function is strictly increasing, meaning its partial derivative is positive.

- Second, we show that when $z_i > z_i^{gt}$, the function is strictly decreasing, meaning its partial derivative is negative.

Proof 8. *We restrict our analysis to the open regions where $IoU_{b_{gt}}(b)$ is infinitely differentiable (i.e., $z_k \neq z_k^{gt}$ for all k), as stated in Property 5.2. We focus on the partial derivative with respect to z_0. The identical logic applies to the other coordinates (z_1, z_2, z_3). To make the equations easier to read, we define the geometric properties of the bounding boxes. Since we only vary z_0, the heights are constant:*

- *Height of the predicted bounding box: $H_b = z_3 - z_1$*

- *Height of the intersection: $H_{int} = \min(z_3, z_3^{gt}) - \max(z_1, z_1^{gt})$*

Under Hypothesis 5.1, the boxes overlap, meaning $H_b > 0$ and $H_{int} > 0$. Furthermore, the intersection height cannot exceed the prediction height, so $H_b \geq H_{int}$.

We introduce the width of the intersection that depends on z_0 and is defined as $W_{int}(z_0) = \min(z_2, z_2^{gt}) - \max(z_0, z_0^{gt})$.

To simplify our expressions based on Definition 5.2, we rewrite the IoU as a quotient by introducing $d_{gt}(b)$ as the area of the union. Thus, we have:

$$IoU_{b_{gt}}(b) = \frac{a(i_{b_{gt}}(b))}{d_{gt}(b)}$$
$$\text{with } d_{gt}(b) = a(b_{gt}) + a(b) - a\Big(i_{b_{gt}}(b)\Big)$$

We calculate the partial derivative of the IoU using the quotient rule. For any $z_0 \in \mathbb{R}$, we obtain:

$$\frac{\partial IoU_{b_{gt}}(b)}{\partial z_0} = \frac{d_{gt}(b) \cdot \frac{\partial a(i_{b_{gt}}(b))}{\partial z_0} - a(i_{b_{gt}}(b)) \cdot \frac{\partial d_{gt}(b)}{\partial z_0}}{d_{gt}(b)^2} \tag{5.8}$$
$$\text{with } d_{gt}(b) = a(b_{gt}) + a(b) - a\Big(i_{b_{gt}}(b)\Big)$$

We first analyze the case where the coordinate is smaller than the ground truth ($z_0 < z_0^{gt}$) to show that the function is strictly increasing in this region.

Case A: The coordinate is smaller than the ground truth ($z_0 < z_0^{gt}$)

In this region, the maximum function evaluates to the ground truth value: $\max(z_0, z_0^{gt}) = z_0^{gt}$. The width of the intersection is $W_{int} = \min(z_2, z_2^{gt}) - z_0^{gt}$. This width remains constant with respect to z_0. Consequently, the intersection area $a(i_{b_{gt}}(b)) = H_{int} \cdot W_{int}$ is constant with respect to z_0. Its partial derivative is zero:

$$\frac{\partial a(i_{b_{gt}}(b))}{\partial z_0} = 0$$

The area of the predicted bounding box is $a(b) = H_b \cdot (z_2 - z_0)$*. Its partial derivative is:*

$$\frac{\partial a(b)}{\partial z_0} = -H_b \tag{5.9}$$

For the union area $d_{gt}(b) = a(b_{gt}) + a(b) - a(i_{b_{gt}}(b))$*, only* $a(b)$ *depends on* z_0*. Its partial derivative is:*

$$\frac{\partial d_{gt}(b)}{\partial z_0} = -H_b$$

We substitute these terms into our initial Equation (5.8)*:*

$$\frac{\partial IoU_{b_{gt}}(b)}{\partial z_0} = \frac{d_{gt}(b) \cdot 0 - a(i_{b_{gt}}(b)) \cdot (-H_b)}{d_{gt}(b)^2} = \frac{a(i_{b_{gt}}(b)) \cdot H_b}{d_{gt}(b)^2}$$

Since $a(i_{b_{gt}}(b)) > 0$ *and* $H_b > 0$*, the numerator is strictly positive. The denominator is a squared term, making it strictly positive. Therefore, the partial derivative is strictly positive. The function is strictly increasing when* $z_0 < z_0^{gt}$*.*

Case B: The coordinate is larger than the ground truth ($z_0 > z_0^{gt}$ ***)***

In Case A, the width of the intersection W_{int} *was constant with respect to* z_0*. Here, the width of the intersection is* $W_{int}(z_0) = \min(z_2, z_2^{gt}) - z_0$ *does not remain constant with respect to* z_0*. Consequently, both the predicted bounding box area and the intersection area vary with* z_0*, requiring a more detailed calculation for the partial derivative.*

In this region, the maximum function evaluates to z_0*:* $\max(z_0, z_0^{gt}) = z_0$*. The width of the intersection is now* $W_{int}(z_0) = \min(z_2, z_2^{gt}) - z_0$*. The intersection area becomes* $a(i_{b_{gt}}(b)) = H_{int} \cdot (\min(z_2, z_2^{gt}) - z_0)$*. Its partial derivative is:*

$$\frac{\partial a(i_{b_{gt}}(b))}{\partial z_0} = -H_{int}$$

The predicted bounding box area derivative remains the same as Equation (5.9)*:*

$$\frac{\partial a(b)}{\partial z_0} = -H_b$$

For the union area $d_{gt}(b)$*, the partial derivative involves both terms:*

$$\frac{\partial d_{gt}(b)}{\partial z_0} = \frac{\partial a(b)}{\partial z_0} - \frac{\partial a(i_{b_{gt}}(b))}{\partial z_0} = -H_b - (-H_{int}) = -(H_b - H_{int})$$

We substitute these terms into the numerator of Equation (5.8)*:*

$$\begin{aligned} Numerator &= d_{gt}(b) \cdot (-H_{int}) - a(i_{b_{gt}}(b)) \cdot (-(H_b - H_{int})) \\ &= -d_{gt}(b) \cdot H_{int} + a(i_{b_{gt}}(b)) \cdot (H_b - H_{int}) \end{aligned}$$

We replace $d_{gt}(b)$ with its full expression $a(b_{gt}) + a(b) - a(i_{b_{gt}}(b))$:

$$\begin{aligned} Numerator &= -\Big(a(b_{gt}) + a(b) - a(i_{b_{gt}}(b))\Big) \cdot H_{int} + a(i_{b_{gt}}(b)) \cdot H_b - a(i_{b_{gt}}(b)) \cdot H_{int} \\ &= -a(b_{gt}) \cdot H_{int} - a(b) \cdot H_{int} + a(i_{b_{gt}}(b)) \cdot H_{int} + a(i_{b_{gt}}(b)) \cdot H_b - a(i_{b_{gt}}(b)) \cdot H_{int} \\ &= -a(b_{gt}) \cdot H_{int} - a(b) \cdot H_{int} + a(i_{b_{gt}}(b)) \cdot H_b \end{aligned}$$

We write the areas $a(b)$ and $a(i_{b_{gt}}(b))$ using their heights and widths. Let $W_b(z_0) = (z_2 - z_0)$ be the width of the predicted bounding box. We get:

$$\begin{aligned} -a(b) \cdot H_{int} + a(i_{b_{gt}}(b)) \cdot H_b &= -(H_b \cdot W_b(z_0)) \cdot H_{int} + (H_{int} \cdot W_{int}(z_0)) \cdot H_b \\ &= H_b \cdot H_{int} \cdot \Big(W_{int}(z_0) - W_b(z_0)\Big) \end{aligned}$$

Since the intersection width cannot be larger than the predicted bounding box width, we know that $W_{int}(z_0) \leq W_b(z_0)$. This means the difference $(W_{int}(z_0) - W_b(z_0))$ is negative or zero. As a result, the term $H_b \cdot H_{int} \cdot \Big(W_{int}(z_0) - W_b(z_0)\Big)$ is negative or zero. The remaining term in the numerator is $-a(b_{gt}) \cdot H_{int}$, which is strictly negative. Adding a negative term to a negative or zero term guarantees that the numerator is strictly negative.

Therefore, the partial derivative is strictly negative. The function is strictly decreasing when $z_0 > z_0^{gt}$.

Conclusion

Since it strictly increases when $z_0 < z_0^{gt}$ and strictly decreases when $z_0 > z_0^{gt}$, it reaches a unique local maximum at $z_0 = z_0^{gt}$.

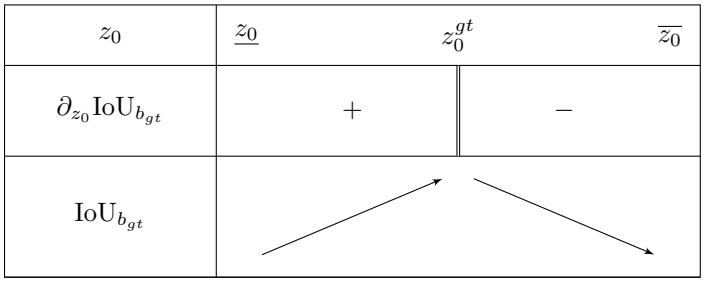


Figure 5.11: Variation table of $\mathrm{IoU}_{b_{gt}}(b)$ with respect to z_0.

This logic applies independently to all coordinates (z_0, z_1, z_2, z_3), proving that $IoU_{b_{gt}}(b)$ is unimodal with respect to each coordinate, we have successfully established Proposition 8. □

5.3.2 Our method: Bounding $\text{IoU}_{b_{gt}}(b)$ using its unimodality in each coordinate

For our method, we use the unimodality of $\text{IoU}_{b_{gt}}(b)$ to verify the robustness Property (Equation (5.6)). As explained in Subsection 5.3.1, to determine the guaranteed minimum $\text{IoU}_{b_{gt}}$ value, i.e. $\min_{b \in f^a(\mathcal{C})} \text{IoU}_{b_{gt}}(b)$, it is sufficient to evaluate the function at the bounds (the vertices) of the input domain. Practically, this property reduces our continuous optimization problem to an evaluation of a discrete set of 16 extreme configurations ($2^4 = 16$). Figure 5.12 illustrates these 16 candidate bounding boxes constructed by selecting either the lower bound ($\underline{z_k}$) or the upper bound ($\overline{z_k}$) for each of the four coordinates.

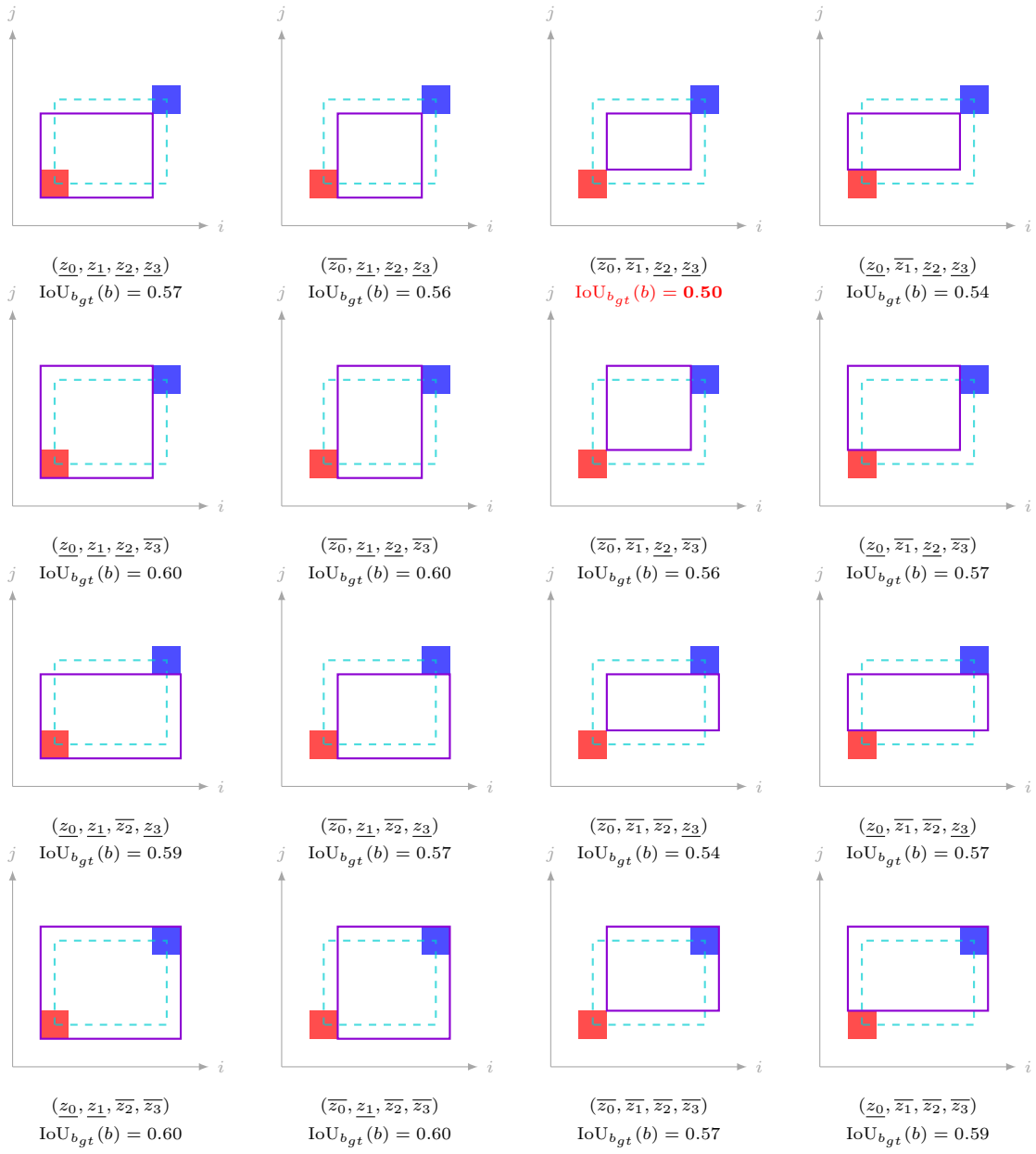


Figure 5.12: Visualization of the 16 bounding box combinations evaluated at the vertices of the interval bounds $f^a(\mathcal{C})$.

In Figure 5.12, the worst bounding box prediction, which is the box in $f^a(\mathcal{C})$ that yields the guaranteed minimum $\text{IoU}_{b_{gt}}$ value, is highlighted in red, such that $\min_{b \in f^a(\mathcal{C})} \text{IoU}_{b_{gt}}(b) = 0.50$. Since this minimum value satisfies the standard safety threshold ($\tau = 0.5$), the robustness property defined in Equation (5.6) is verified.

Based on the interval coordinates $[\underline{z_k}, \overline{z_k}]$ (Equation (5.5)), Proposition 8 guarantees that this method produces the tightest possible minimum and maximum of $\text{IoU}_{b_{gt}}(b)$ for any $b \in f^a(\mathcal{C})$.

Remark 47 (Clarifying the term *exact*)**.** *In formal verification, the word exact usually describes a method that uses no approximations. To prevent any confusion regarding our two-step process (Figure 5.8), we must clarify the following:*

- ***Step 1 is an approximation:*** *The interval bounds $[\underline{z_k}, \overline{z_k}]$ computed by verification tools like CROWN are an over-approximation of the neural network's reachable outputs.*
- ***Step 2 is an exact calculation:*** *Our proposed method finds the exact guaranteed minimum of the $IoU_{b_{gt}}(b)$ metric within these specific intervals, without adding any new approximations.*

Consequently, while Step 2 computes the tightest possible minimum for the given intervals, the overall robustness verification remains an over-approximation (due to Step 1).

5.3.3 Algorithmic implementation and code

To evaluate the robustness of the neural network f, we use the unimodality property to determine the guaranteed minimum $\text{IoU}_{b_{gt}}$ value (global). This procedure consists of the following steps:

- **Step 1**:
  - **Extract bounds (Step 1):** Obtain the interval bounds $[\underline{z_k}, \overline{z_k}]$ for all four coordinates $k \in \{0, \ldots, 3\}$ from the neural network over-approximation $f^a(\mathcal{C})$. We rely on `auto_LiRPA` using the IBP, CROWN, or CROWN-IBP methods. As shown in Listing 2.2 (Subsection 2.4), `auto_lirpa` provides these lower and upper bounds as Pytorch tensors output of the `compute_bounds` function.

    Remark 48 (Automatic concretization)**.** *As established in Remark 46, our approach requires interval coordinates $[\underline{z_k}, \overline{z_k}]$. This means that any linear bounds computed by verification methods must be concretized, i.e., transformed into intervals (see Remark 10). While methods like CROWN and CROWN-IBP compute linear bounds, `auto_LiRPA` automatically performs this concretization by default. If necessary, a specific mode can be enabled to retrieve the linear bounds.*
- **Step 2**:
  - **Construct vertices :** Generate the 16 candidate bounding boxes by taking every possible combination of lower and upper bounds, as illustrated in Figure 5.12.
  - **Compute scores:** Calculate the $\text{IoU}_{b_{gt}}$ value for each of the 16 bounding boxes using Equation (5.4).
  - **Find the minimum:** Select the smallest score among these 16 computed values. This specific value represents the guaranteed minimum $\text{IoU}_{b_{gt}}$ value, $\min_{b \in f^a(\mathcal{C})} \text{IoU}_{b_{gt}}(b)$.

– **Verify robustness:** Compare this minimum value to the predefined safety threshold τ. If the minimum value is greater than or equal to τ, we successfully prove that every possible bounding box in the over-approximation $f^a(\mathcal{C})$ is correct. Under these conditions, the object detection neural network is verified as robust against the tested perturbations.

The step 2 is encoded into Pytorch tensor operations.

We provide an open-source implementation[2].

Remark 49 (Identifying the worst bounding box prediction)**.** *By tracking the index of the minimum $IoU_{b_{gt}}$ value during the vertex evaluation, we can directly recover the specific combination of coordinates that represents the worst bounding box prediction.*

In conclusion, this approach guarantees the minimum $\text{IoU}_{b_{gt}}(b)$ value by testing only 16 discrete points. This discrete evaluation provides a mathematical proof of the neural network robustness for the specified input bounds. For clarity in the following experiments, we refer to the method as *Unimodal-IoU*. We now explore the Experiments.

5.4 Experiment

This section presents the experimental evaluation of *Unimodal-IoU*. We first describe the experimental setup, including the datasets, neural networks and perturbations and then the results.

5.4.1 Experimental Setup

The verification approach is tested on two object detection use cases and a number of perturbations. We focus on convolutional object detection neural networks. All experiments are in part parallelized over a pool of 20 workers, on a Linux machine with Intel® Xeon® processor E5-2660 v3 @ 2.60GHz of 20 cores and 64 GB RAM. We explore two datasets, namely:

- **DIGIT_LOC**: MNIST LeCun et al. [2002] pictures of size 28×28 are randomly placed on black pictures of size of 90×90, thus the ground truth bounding box has a fixed size on all pictures and its coordinates correspond to the position of the MNIST picture. The generated dataset is uniformly composed of pictures representing digits from 0 to 9. The verification is conducted on 40 pictures.
- **LARD**: LARD dataset Ducoffe et al. [2023] comprises high-quality aerial pictures of runway during approach and landing phases. We select 40 synthetic pictures from the Reykjavík domestic airport taken into clear weather conditions within a distance range of 0.33 to 1.08 nautical miles (NM) from the runway. The original pictures are $3 \times 2448 \times 2648$ pixels. We resize them to $3 \times 256 \times 256$ pixels. After resizing, the runway areas range from 70 to 706 square pixels. This second use case is more challenging for robustness verification due to varying ground truth bounding box sizes.

We normalize pixel values to a scale ranging from 0 to 1. We train two neural network for each dataset. Both of them return four values that predicts the four coordinates

[2] https://github.com/NoCohen66/Verification4ObjectDetection

(see Remark 42). We describe the network architectures for each one, representing a convolutional layer as a 4-tuple consisting of (number of filters, kernel size, stride, and padding):

- **DIGIT_LOC:** 1 conv layer $\{(16, 3, 1, 1)\}$, 1 pooling layer with 2×2 kernel size and stride 2, 1 conv layer $\{(16, 3, 1, 1)\}$, and 1 pooling layer with 2×2 kernel size and stride 2, followed by 2 linear layers with $\{256, 4\}$ neurons. All layers are followed by a ReLU activation, except for the final output layer.
- **LARD:** 3 conv layers $\{(32, 3, 2, 1), (64, 3, 2, 1), (128, 3, 2, 1)\}$ followed by 3 linear layers with $\{128, 128, 4\}$ neurons. All layers are followed by a ReLU activation, except for the final output layer.

While we concentrate on convolutional neural network, Nirala and Sarkar [2025] focuses YOLO neural networks.

5.4.2 Metrics and perturbations

We introduce the notion of *Certified accuracy (%)* for object detection that corresponds to the fraction of test pictures fulfilling the robustness guarantee property from Equation (5.6) with a threshold $t = 0.5$. For instance, a certified accuracy of 10% means that only 10% of the 40 evaluated pictures (i.e., 4 pictures) satisfy the property. We evaluate the robustness of the neural networks against three types of picture perturbations:

- White-noise, denoted by ϵ (Example 5, Page 27).
- Brightness, denoted by α_b (Example 6, Page 28).
- Contrast, denoted by α_c (Remark 21, Page 51).

Table 5.1, in the second column, summarizes the definitions of these input domains and the specific range of perturbations used in our experiments.

Perturbation	**Definition**	**Param.**	**DIGIT_LOC**			**LARD**		
			Min	Max	Steps	Min	Max	Steps
White-noise	$\{x \in \mathbb{R}^{k_0} \mid \|x - x^\star\|_\infty \leq \epsilon\}$	ϵ	0	0.002	11	0	0.002	11
Brightness	$\{x \in \mathbb{R}^{k_0} \mid x = x^\star + \alpha_b v\}$	α_b	0	0.002	11	0	0.020	11
Contrast	$\{x \in \mathbb{R}^{k_0} \mid x = \alpha_c x^\star\}$	α_c	0	0.200	11	0	0.100	11

Table 5.1: Overview of the input domains around the reference input $x^\star \in \mathbb{R}^{k_0}$ and their respective range of perturbations across the two datasets. v represents a vector of ones.

We evaluate the neural networks across several range of perturbations defined by a minimum value, a maximum value, and a total number of steps. To illustrate this logic, let γ represent any of the perturbation parameters (ϵ, α_b, or α_c). If we configure a minimum 0, a maximum 1, and 4 steps, it yields:

- $-0.25 \leq \gamma \leq 0.25$
- $-0.50 \leq \gamma \leq 0.50$
- $-0.75 \leq \gamma \leq 0.75$
- $-1 \leq \gamma \leq 1$

To reduce space, we write these intervals using the absolute value notation: $|\gamma| \leq 0.25$, $|\gamma| \leq 0.50$, $|\gamma| \leq 0.75$, and $|\gamma| \leq 1$.

Remark 50 (Interval size - not used here)**.** *Unlike the Super-DeepG experiments, this section does not use the Interval Size hyperparameter to split ranges for higher precision (see Subsection 4.4.2, Page 115).*

We use 11 steps for the perturbations mentioned above (ϵ, α_b, and α_c). For the white-noise scenario, we start at $\epsilon = 0$. We then increase the range of perturbations up to a maximum bound of 0.002. This creates perturbation domains such as $|\epsilon| \leq 0$, $|\epsilon| \leq 0.0002$, $|\epsilon| \leq 0.0004$, and so on. We adapt the maximum bounds for each transformation to account for the relative difficulty of the DIGIT_LOC and LARD datasets.

5.4.3 Results

We observe how the $\text{IoU}_{b_{gt}}$ changes as the perturbation increases. Next, Figure 5.13 shows the comparison between the two methods using the Certified accuracy (%) metric under different configurations.

	DIGIT_LOC			LARD		
White noise $\lvert\epsilon\rvert \leq$	0.0002	0.0004	0.0006	0.0004	0.0006	0.0008
CROWN-IBP	100.0	35.9	2.6	8.3	0.0	0.0
CROWN	100.0	66.6	7.70	97.2	75.0	27.8
Brightness $\lvert\alpha_b\rvert \leq$	0.0002	0.0004	0.001	0.004	0.006	0.008
CROWN-IBP	100.0	35.0	0.0	0.0	0.0	0.0
CROWN	100.0	100.0	92.3	94.4	86.1	66.7
Contrast $\lvert\alpha_c\rvert \leq$	0.0002	0.001	0.0014	0.01	0.02	0.03
CROWN-IBP	82.9	0.0	0.0	0.0	0.0	0.0
CROWN	100.0	88.6	8.6	82.5	58.8	38.1

Figure 5.13: Examples of certified accuracy (in %) obtained for some of the tested perturbations for different verification approach combinations and the two use cases.

Based on Figure 5.13, we make the following observations:

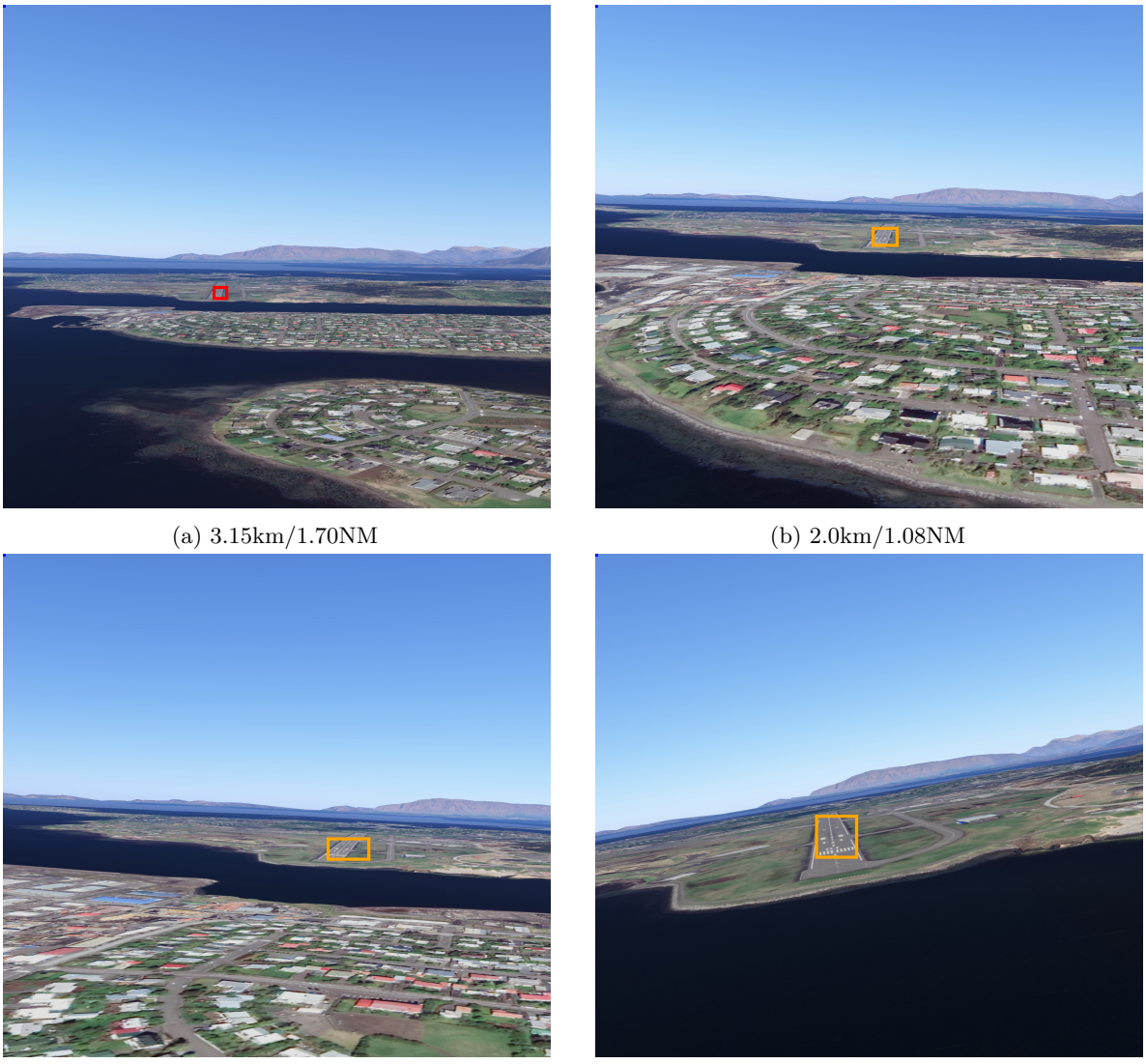
- **The precision of the method depends on the bounds $[\underline{z}_k, \overline{z}_k]$ (Equation (5.5)):** While we expected CROWN to outperform CROWN-IBP as noted in Remark 16, the experimental results reveal an important gap. For example, on the DIGIT_LOC dataset with a noise level of $|\epsilon| \leq 4 \times 10^{-4}$, we find a certified accuracy of 35.9% for CROWN-IBP versus 66.6% for CROWN. We also note that using a pure IBP approach always yields a certified accuracy of 0, meaning it fails to establish any robustness guarantees.

- **The impact of the range of perturbations on robustness:** The results show a clear trend across all experiments: as the range of perturbations increases, the certified accuracy drops. For example, when applying the CROWN method on the DIGIT_LOC dataset, the certified accuracy falls from 100.0% to 7.7% as the white-noise range of perturbations $|\epsilon|$ grows from 0.0002 to 0.0006. This confirms that larger perturbations make it much harder to guarantee the correctness of the neural network for every tested picture. As the input domain grows, it produces worse bounding box predictions that fail to satisfy the robustness property.

A special focus on the LARD use case:

The experiments reveal that the robustness of the neural network depends on the size of the runway in the picture (which correlates with the distance of the plane to the runway). Pictures with smaller runways show greater vulnerability to perturbations. Figure 5.14 highlights four pictures from one landing approach, evaluating robustness against a brightness input domain of $|\alpha_b| \leq 0.002$.

This dependency makes sense. A small perturbation on an object with few pixels creates a larger change in the $\text{IoU}_{b_{gt}}$ calculation than the same perturbation on a large object. This highlights an extra challenge for AI practitioners: training a neural network on a wide range of object sizes requires additional care to ensure robustness across all possible sizes.



(a) 3.15km/1.70NM

(b) 2.0km/1.08NM

(c) 1.63km/0.88NM

(d) 0.78km/0.42NM

Figure 5.14: Impact of a brightness perturbation $|\alpha_b| \leq 0.002$ on a trajectory computed with the CROWN method: orange boxes indicate slight impact, while the red box indicates a strong one (with a guaranteed minimum $\text{IoU}_{b_{gt}}$ value of 0.34).

5.5 Related Work

We categorize the literature on object detection robustness into two main areas.

The first area explores probabilistic approaches. Liu et al. [2026] propose a probabilistic tool to handle the scale of modern object detection neural networks. As this falls outside the scope of deterministic guarantees, we do not cover it further.

The second area focuses on formal verification, which directly concerns our work. Within this area, there are two significant contributions. We and Raviv et al. [2024]

are the first to investigate the formal verification of object detection localization with $\text{IoU}_{b_{gt}}(b)$. Furthermore, since our method achieved better results, recent research by Brückner et al. [2026] builds upon our foundation. In fact, later, Brückner et al. extended our method to anchor-based architectures. We dedicate the following sections to detailing these two approaches.

5.5.1 Raviv et al.

Raviv et al. [2024] presents a state-of-the-art method for the formal verification of object detection. They decompose $\text{IoU}_{b_{gt}}(b)$ on a sequence of primitive mathematical operators.

As we established in Definition 5.2, $\text{IoU}_{b_{gt}}(b)$ (Page 125) relies on a rational function. Bounding a division of two variables introduces significant over-approximation when applying relaxation techniques. To resolve this issue, Raviv et al. [2024] introduce the $\text{CoIoU}_{b_{gt}}(b)$ formulation. This approach eliminates the division operator by setting a fixed correctness threshold, τ, and reformulating the condition.

First, we detail how they mathematically define the $\text{CoIoU}_{b_{gt}}$ function. Next, we explain the step-by-step algorithm they developed to compute it. Finally, we show how they encode this algorithm into `auto-LiRPA`.

5.5.1.1 Mathematical foundations

To address the challenge of bounding a rational function, the method presented by Raviv et al. [2024] introduces a reformulated metric that avoids division. The correctness condition for a single predicted bounding box b is reformulated. According to Definition 5.1, the prediction b is correct if its $\text{IoU}_{b_{gt}}(b)$ meets or exceeds the safety threshold τ:

$$\frac{a(i_{b_{gt}}(b))}{a(b_{gt}) + a(b) - a(i_{b_{gt}}(b))} \geq \tau$$

The denominator of this fraction represents the union area. Under Hypothesis 5.1, this area is strictly positive. We can multiply both sides of the inequality by the denominator without changing the inequality sign:

$$a(i_{b_{gt}}(b)) \geq \tau \cdot \Big(a(b_{gt}) + a(b) - a(i_{b_{gt}}(b))\Big) \quad \Leftrightarrow \quad a(i_{b_{gt}}(b)) - \tau \cdot \Big(a(b_{gt}) + a(b) - a(i_{b_{gt}}(b))\Big) \geq 0$$

This resulting expression forms the mathematical foundation for the CoIoU metric, which we formally define below.

Definition 5.3 (CoIoU). *Let $b = [z_0, z_1, z_2, z_3]$ be a predicted bounding box and $b_{gt} = [z_0^{gt}, z_1^{gt}, z_2^{gt}, z_3^{gt}]$ be a fixed ground truth bounding box. The CoIoU is a function mapping $(\mathbb{R}^4)^2 \longmapsto \mathbb{R}$, defined as:*

$$CoIoU_{b_{gt}}(b) = a(i_{b_{gt}}(b)) - \tau \cdot \Big(a(b_{gt}) + a(b) - a(i_{b_{gt}}(b))\Big) \tag{5.10}$$

We can expand the terms in Equation (5.10) by grouping the intersection area:

$$CoIoU_{b_{gt}}(b) = (1 + \tau) \cdot a(i_{b_{gt}}(b)) - \tau \cdot a(b) - \tau \cdot a(b_{gt}) \tag{5.11}$$

To verify the robustness property for the entire neural network, every possible bounding box inside the over-approximation $f^a(\mathcal{C})$ must satisfy the correctness condition from Equation (5.7) (Page 127). It is equivalent to prove that the minimum CoIoU score over the entire over-approximation is greater than or equal to zero:

$$\left(\min_{b \in f^a(\mathcal{C})} \text{IoU}_{b_{gt}}(b)\right) \geq \tau \quad \Leftrightarrow \quad \left(\min_{b \in f^a(\mathcal{C})} \text{CoIoU}_{b_{gt}}(b)\right) \geq 0 \tag{5.12}$$

Finally, Raviv et al. [2024] address the same problem as *Unimodal-IoU*.

5.5.1.2 Decomposition per operator

The $\text{CoIoU}_{b_{gt}}$ formula involved operators: minimum (min), maximum (max), addition ($+$), subtraction ($-$), multiplication ($\times$). The function requires two main inputs: the predicted bounding box coordinates and the area of the ground truth bounding box, $a(b_{gt})$. For a given picture, this ground truth area is a constant value (Remark 45). Using these inputs, the function calculates and outputs the final $\text{CoIoU}_{b_{gt}}$ score. Figure 5.15 visualizes the decomposition of the mathematical operators involved in the $\text{CoIoU}_{b_{gt}}$ formula.

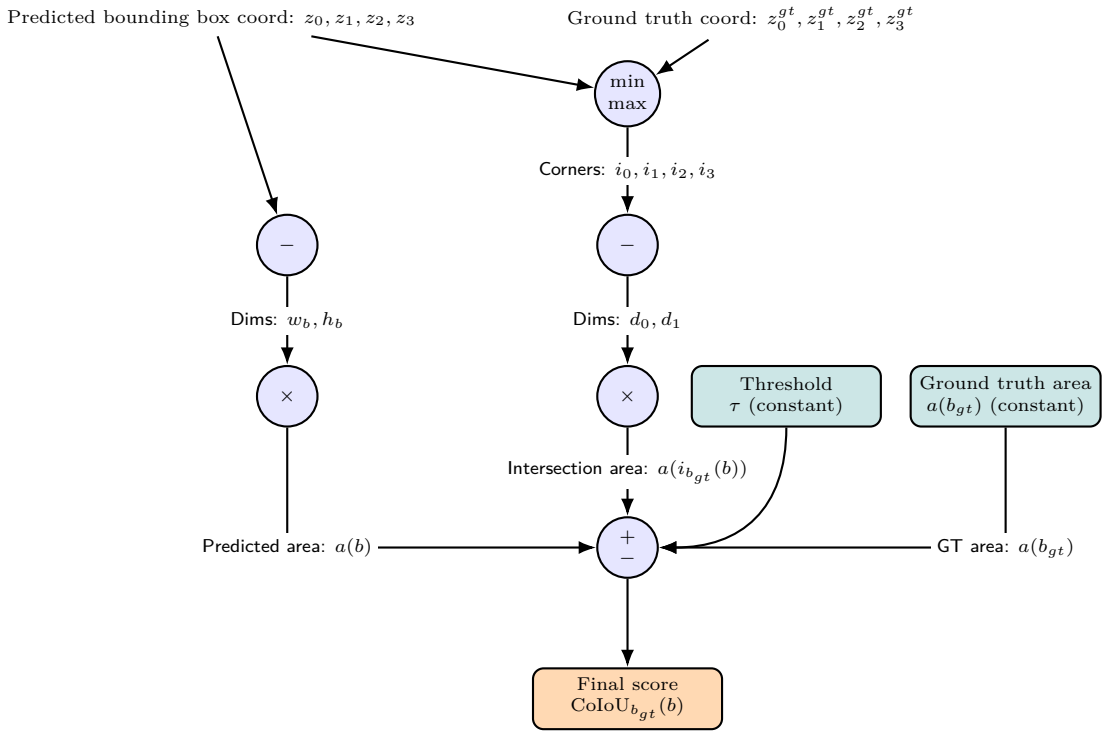


Figure 5.15: Computational graph for the $\text{CoIoU}_{b_{gt}}(b)$ metric.

5.5.1.3 Encoding operators as neural network layers

The authors of Raviv et al. [2024] observe that every mathematical node in the $\text{CoIoU}_{b_{gt}}$ graph (Figure 5.15) corresponds to an operation supported by the `auto-LiRPA` solver. Instead of computing the metric as a separate post-processing step, they formulate these operators as a sequence of neural network layers.

We can view each operator as a *wrapper layer*, similar to Example 10. The sequence of operations for $\text{CoIoU}_{b_{gt}}$ can be considered a neural network. Appending this wrapper network $f_{wrapper}$ to the original object detector f creates a unified composite neural network, denoted by $f_{wrapper} \circ f$.

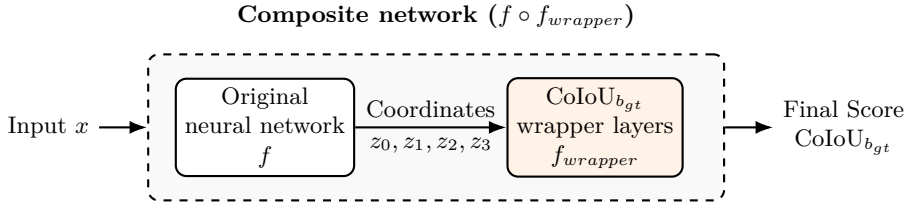


Figure 5.16: Composition of $\text{CoIoU}_{b_{gt}}$ wrapper layers with the neural network.

This strategy allows CROWN (or CROWN-IBP) methods to compose the linear bounds of the original network f with $f_{wrapper}$ using the backward composition rule (Property 2.6). Because of this composition, the bounding box coordinates (the output of f) preserve their linear relationship with the initial input perturbation $\kappa \in \mathbb{R}^{d\times 1}$. Let $\underline{c_k}, \overline{c_k} \in \mathbb{R}^{1\times d}$ and $\underline{b_k}, \overline{b_k} \in \mathbb{R}$ represent these linear coefficients. The over-approximation is defined by:

$$f^a(\mathcal{C}) = \left\{ b \in \mathbb{R}^4 \;\middle|\; \underline{c_k} \cdot \kappa + \underline{b_k} \leq z_k \leq \overline{c_k} \cdot \kappa + \overline{b_k}, \quad \forall k \in \{0, \ldots, 3\} \right\} \tag{5.13}$$

Ultimately, the CROWN composition yields a final linear relaxation of the $\text{CoIoU}_{b_{gt}}$ metric in terms of the initial input perturbation κ. With $\underline{a}, \overline{a} \in \mathbb{R}^{1\times d}$ and $\underline{c}, \overline{c} \in \mathbb{R}$ as the final linear coefficients, the solver achieves the bound:

$$\underline{a} \cdot \kappa + \underline{c} \leq \text{CoIoU}_{b_{gt}} \leq \overline{a} \cdot \kappa + \overline{c} \tag{5.14}$$

However, this method still introduces some over-approximations.

Property 5.5 (Over-approximation of non-linear operators)**.** *While the linear operations are exact under linear relaxation, the non-linear ReLU layers used to encode the* max *and* min *functions introduce approximations. As illustrated previously in Figure 2.12, relaxing a ReLU function creates an over-approximation area. Thus, the final bounds for the $CoIoU_{b_{gt}}$ score remain an over-approximation of the true reachable set.*

To obtain the final linear relaxation bounds in Equation (5.14), the methodology relies on three algorithmic steps:

- **Step a:** Decompose the $\text{CoIoU}_{b_{gt}}$ formulation into a sequence of mathematical operations to create a wrapper neural network, denoted as $f_{wrapper}$.
- **Step b:** Convert these mathematical operators into specific bounding functions.
- **Step c:** Execute a backward bound propagation through this entire graph using the CROWN or CROWN-IBP algorithm to compute the linear coefficients.

In practice, the `auto-LiRPA` framework handles the Steps b and c automatically:

- **Step b:** `auto-LiRPA` converts these standard PyTorch functions into specific `Bound` nodes (such as `BoundMax`, `BoundRelu`, and `BoundMul`).
- **Step c:** Then it applies the relaxation methods using `compute_bounds` (Listing 2.2).

Therefore, to apply this verification method, one only needs to implement the $\text{CoIoU}_{b_{gt}}$ metric as a standard PyTorch network (**Step a**).

5.5.1.4 Experimental comparison

Encoding the metric as a wrapper layer is convenient for practitioners, but this approach accumulates over-approximation (Property 5.5). Each relaxed mathematical operator, such as min and max, contributes to this error and produces loose bounds. Table 5.2 presents the comparison between the two approaches. We extracted the the table from their publication Raviv et al. [2024].

Unimodal-IoU method verifies more robust test pictures on both datasets: 240 versus 183 for MNIST-OD, and 263 versus 135 for LARD. In addition, our method runs faster (248× and 146× faster). In means that the over-approximation from concretizing our interval bounds (Equation (5.5)) is strictly smaller than the over-approximation from relaxing the non-linear operators.

While Raviv et al. focused on encoding the metric as a neural network layer, other researchers focus on other architectures.

Method	MNIST-OD		LARD	
	Robust	**Time (s)**	**Robust**	**Time (s)**
`Unimodal-IoU` (Ours)	**240**	**0.011**	**263**	**0.011**
Raviv et al.	183	2.731	135	1.606

Table 5.2: Comparison of verification performance between our approach (`Unimodal-IoU`) and Raviv et al on white-noise perturbations. *Robust* corresponds to the number of test pictures fulfilling the robustness guarantee property from Equation (5.6) with a threshold $t = 0.5$.

5.5.2 Brückner et al.

Recent work by Brückner et al., introducing the IoUCert method, addresses robustness verification for anchor-based object detection neural network.

5.5.2.1 Verification problem

To clarify the differences between their approach and ours, we compare them across several key aspects:

- **Inference:**
  - **Our architecture:** In our methodology, we focus on convolutional neural networks that output the bounding box coordinates $b = (z_0, z_1, z_2, z_3)$. Since this type of neural network is not very common in practice, we trained these custom neural networks ourselves for our experiments. As stated in Remark 42, we restrict our scope to single object detection by selecting the bounding box with the highest confidence score. Figure 5.17 a) illustrates the inference.
  - **Their architecture:** Brückner et al. target realistic architectures, such as YOLO and SSD. These neural networks do not output coordinates. Instead, they output offsets $o = (o_0, o_1, o_2, o_3)$ relative to predefined reference boxes called anchors. The final bounding box $b = (z_0, z_1, z_2, z_3)$ is calculated using a composition of functions h and ϕ, such that $b = h \circ \phi(o)$[3]. As a result, the bounding box coordinates z_0, z_1, z_2, z_3 are constrained by the predicted offsets. They restrict their scope to single object detection by selecting the offset with the highest confidence score. Figure 5.17 b) illustrates this inference workflow.

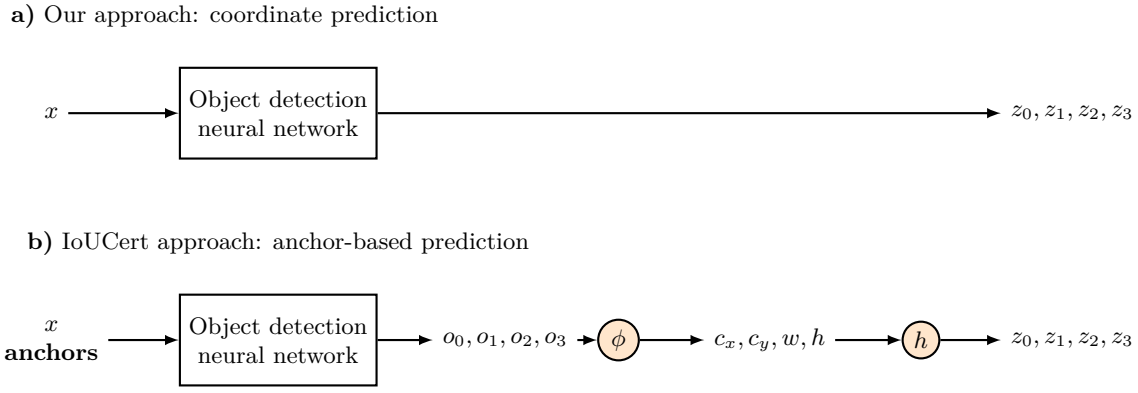


Figure 5.17: Comparison of the inference workflow between coordinate and anchor-based predictions.

[3]The post-processing function ϕ transforms offsets into center coordinates (c_i, c_j, w, h) where (c_i, c_j) are the center coordinates, w the width, and h the height. Following this step, the function h converts the center coordinates into bounding box coordinates z_0, z_1, z_2, z_3.

- **Verification:**
  - **Our verification workflow:** In our approach, there are two steps. The first step, the relaxation of the neural network, produces a set of over-approximated bounding boxes (see Figure 5.5, Page 127). The second step compares the guaranteed minimum $\text{IoU}_{b_{gt}}$ value to the safety threshold τ. Figure 5.18 a) illustrates our verification workflow.
  - **Their verification workflow:** In their approach, there are also two steps. The relaxation of the neural network (with the `VENUS` solver), produces a set of over-approximated **offsets**. Then, they calculate the guaranteed minimum $\text{IoU}_{b_{gt}}$ value based on these offsets and the post-processing functions h and ϕ. The verification problem for Step 2 is:

$$\left(\min_{o \in f^a(\mathcal{C})} \text{IoU}_{b_{gt}}(h \circ \phi(o)) \right) \geq \tau \tag{5.15}$$

    Figure 5.18 b) illustrates their verification workflow.

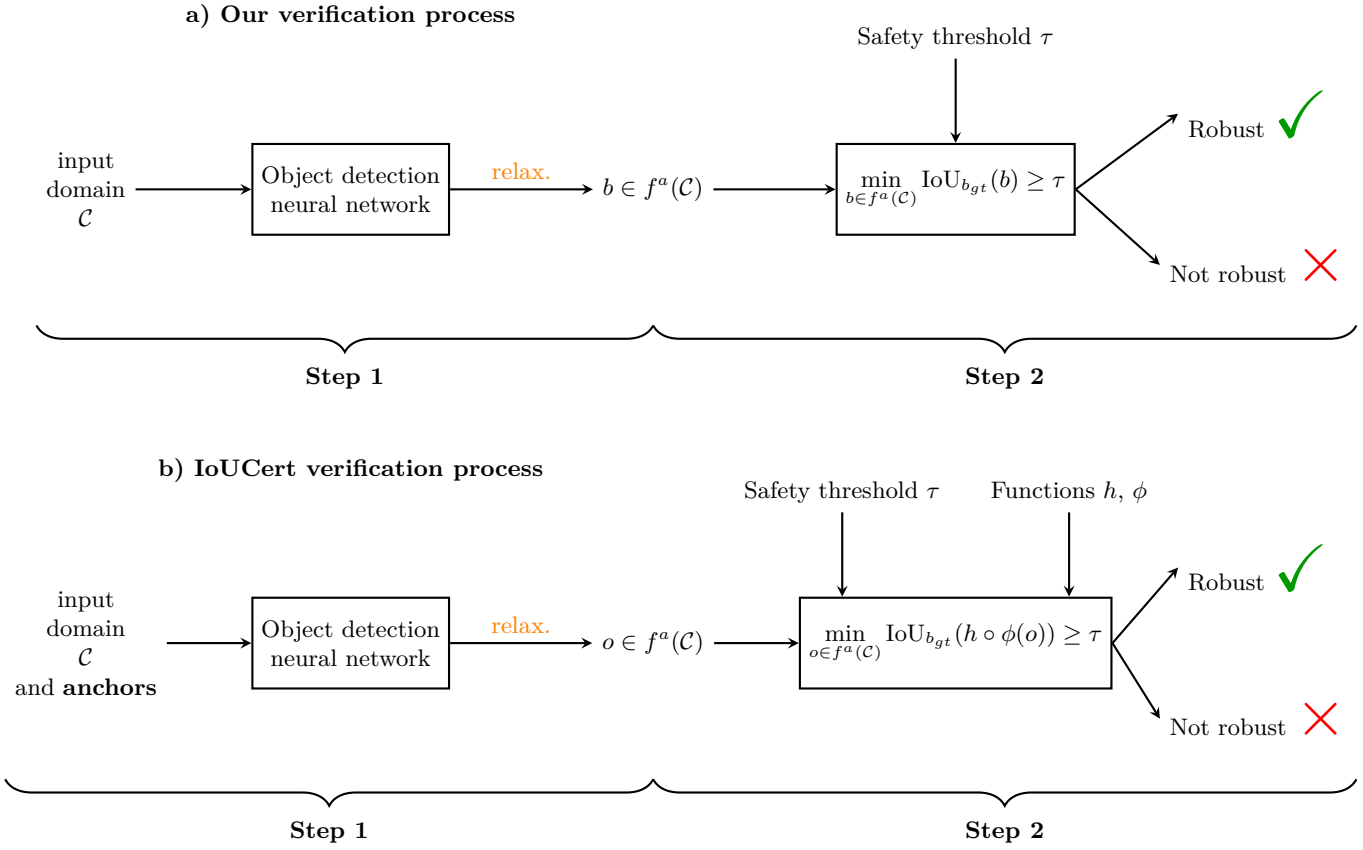


Figure 5.18: Comparison of the verification steps.

- **Solving the Step 2 verification problem:**
  - **Our resolution:** In our approach, the bounding box coordinates z_0, z_1, z_2, z_3 are constrained independently (e.g., $\underline{z_0} \leq z_0 \leq \overline{z_0}$, see Equation (5.5), Page 126). We prove the $\text{IoU}_{b_{gt}}$ function is unimodal with respect to each coordinate independently, then we need to evaluate 16 combinations of coordinates to find the guaranteed minimum $\text{IoU}_{b_{gt}}$ value.

– **Their resolution:** However, their problem includes additional mathematical constraints between the variables. For instance, one of the constraint is $2c_x(\underline{o}_0) \leq z_0 + z_2 \leq 2c_x(\overline{o}_0)$, it links the variation of z_0 and z_2. This coupling increases the complexity, resulting in 169 combinations of coordinates to evaluate in order to find the guaranteed minimum $\text{IoU}_{b_{gt}}$ value.

Remark 51 (Non Max Suppression (NMS))**.** *Both their method and ours leave the complexity of multi-object detection with Non Max Suppression (NMS) for future work.*

The following section details their verification problem and their resolution.

5.5.2.2 Details on the verification problem

The inputs of the verification are the input domain $\mathcal{C}$ and anchors. The goal is to calculate the guaranteed minimum $\text{IoU}_{b_{gt}}$ value to verify the robustness property in Equation (5.15).

The relaxation of the object detection neural network $f^a(\mathcal{C})$, the output of Step 1, is a set of over-approximated offsets. They concretize them to obtain interval bounds for each offset[45]:

$$f^a(\mathcal{C}) = \left\{ o \in \mathbb{R}^4 \;\middle|\; \underline{o_k} \leq o_k \leq \overline{o_k}, \quad \forall k \in 0, \ldots, 3 \right\}$$

Notation 7. *The lower and upper bounds of these predicted offsets within the over-approximation are denoted* $\underline{o} = (\underline{o_0}, \underline{o_1}, \underline{o_2}, \underline{o_3})$ *and* $\overline{o} = (\overline{o_0}, \overline{o_1}, \overline{o_2}, \overline{o_3})$.

To check the robustness property in Equation (5.15) they formulate the following optimization problem:

$$\begin{aligned} &\min_{o} \text{IoU}_{b_{gt}}(h \circ \phi(o)) \\ &\text{such that } \; \underline{o} \leq o \leq \overline{o} \end{aligned}$$

Propagating interval bounds through the non-linear transformation $h \circ \phi$ introduces over-approximation. To avoid this issue, the authors prove that both h and ϕ are injective functions. This mathematical property allows them to use the inverse of $b = h \circ \phi(o)$, i.e. $o = \phi^{-1} \circ h^{-1}(b)$. Then the optimization problem is:

$$\begin{aligned} &\min_{b} \text{IoU}_{b_{gt}}(b) \\ &\text{such that } \; \underline{o} \leq \phi^{-1} \circ h^{-1}(b) \leq \overline{o} \end{aligned}$$

Since ϕ is monotonically increasing, they can apply it to the inequality without changing the direction of the inequality:

$$\begin{aligned} &\min_{b} \text{IoU}_{b_{gt}}(b) \\ &\text{such that } \; \phi(\underline{o}) \leq h^{-1}(b) \leq \phi(\overline{o}) \end{aligned} \tag{5.16}$$

By replacing $h^{-1}(b)$ and ϕ in Equation (5.16), they obtain the final constrained optimization problem:

$$\begin{aligned} \min_{b} \quad & \text{IoU}_{b_{gt}}(b) \\ \text{such that} \quad & 2c_x(\underline{o}_0) \leq z_0 + z_2 \leq 2c_x(\overline{o}_0) \\ & 2c_y(\underline{o}_1) \leq z_1 + z_3 \leq 2c_y(\overline{o}_1) \\ & w(\underline{o}_2) \leq z_2 - z_0 \leq w(\overline{o}_2) \\ & h(\underline{o}_3) \leq z_3 - z_1 \leq h(\overline{o}_3) \end{aligned} \tag{5.17}$$

[4]This is why they refer to *IBP* in their work.

[5]We also concretize the bounds of the bounding box coordinates (Remark 46, Page 128)

Remark 52 (Couple variables)**.** *In our approach, the bounding box coordinates z_0, z_1, z_2, z_3 are constrained independently (e.g., $\underline{z_0} \leq z_0 \leq \overline{z_0}$). However, the constraints in Equation* (5.17) *couple the variables. For instance, the constraint $2c_x(\underline{o}_0) \leq z_0 + z_2 \leq 2c_x(\overline{o}_0)$ links the variation of z_0 and z_2.*

Brückner et al. solve the optimization problem presented in Equation (5.17). To do so, they follow standard optimization principles Boyd and Vandenberghe [2004]. They use the notion of the feasible region, defined in Definition 5.4.

Definition 5.4 (Feasible region)**.** *The feasible region is the set of all variables that satisfy every constraint of the optimization problem.*

The solution of the optimization problem in Equation (5.17) must be located either (i) where the gradient of the function is zero, (ii) on the boundaries of the feasible region, or (iii) at points where the function is non-differentiable. They reduce the search space:

- **(i) Analyzing gradient of the function.** First, the authors reuse our results from Section 5.3.1.2, page 132 : we demonstrated that the partial derivatives of the $\text{IoU}_{b_{gt}}$ function are never zero within the feasible region. Therefore, they only consider cases (ii) and (iii).
- **(ii) Analyzing the diagonal boundaries.** In their case, the boundaries of their feasible region are diagonal lines (e.g., defined by $z_0 + z_2 = \text{constant}$). They analyze the partial derivative of the $\text{IoU}_{b_{gt}}$ function along these diagonal constraints (detailed in their Appendix F.2). They demonstrate that the guaranteed minimum $\text{IoU}_{b_{gt}}$ is at the corners of the feasible region.

Finally, they evaluate the $\text{IoU}_{b_{gt}}$ at:

- (ii) The corners of the feasible region.
- (iii) The points of non-differentiability, which occur where the ground truth coordinates intersect the boundary constraints (e.g., $z_0 = z_0^{gt}$).

Finally, among those 169 values, they identify the minimum. This minimum value is the guaranteed minimum $\text{IoU}_{b_{gt}}$ value.

To conclude, Brückner et al. address the robustness verification problem by focusing on a different neural network architecture. They define a new optimization problem tailored specifically for anchor-based neural networks. This approach successfully extends formal robustness verification to more complex architectures, such as SSD and YOLO.

Having positioned our work within the current literature, we now summarize our main contributions and findings in the conclusion of this chapter.

5.6 Conclusion

In this chapter, we extended robustness verification from classification to object detection. Our goal was to guarantee that an object detection neural network f predicts a correct bounding box b even when the input picture faces perturbations: brightness, contrast and white-noise. To measure this correctness, we relied on the Intersection over Union metric, denoted as $\text{IoU}_{b_{gt}}(b)$. We established a mathematical framework to find the guaranteed minimum of this metric over a set of relaxed bounding boxes, $f^a(\mathcal{C})$.

We analyzed the mathematical properties of the $\text{IoU}_{b_{gt}}(b)$ formula. We demonstrated that the metric is neither convex nor concave, but it is unimodal with respect to

each bounding box coordinate. Our *Unimodal-IoU* method identifies the guaranteed minimum score by testing 16 discrete vertices (the boundaries of the input intervals). This approach avoids the over-approximations introduced by non-linear operators (as Raviv et al. [2024]). Brückner et al. [2026] extended our method to anchor-based architectures.

Part III

Conclusion and Future Work

Chapter 6

Conclusion

Given the promising use of neural networks in *safety-critical systems* (e.g., avionics), this PhD proposed methods and tools to assess some safety properties of such systems. This work has been conducted in parallel with initiatives from both industrial actors and regulatory agencies in aviation, who are engaged in addressing neural network certification issues. Integrating neural networks into safety-critical systems necessitates a new paradigm in system function development due to their data-driven nature. The European Union Aviation Safety Agency (EASA) has released Artificial Intelligence Concept Papers (Issues 1 and 2, with Issue 3 forthcoming) to provide a roadmap and guidelines for neural networks in aeronautics. The working group EUROCAE WG-114 is currently preparing technical standards for the certification of aeronautical systems incorporating neural network components. These documents have guided our work which, in turn, aim to support these initiatives.

6.1 The challenge of high-dimensional robustness

To recall the motivation from the Introduction (Chapter 1), traditional avionics software relies on a requirement-based development process known as the V-model. Neural networks disrupt this established process because they are data-driven. The new W-shape development cycle (Figure 1.5, Page 15) requires a verification step to prove that the designed ML model satisfies its robustness[1] requirements. The robustness property guarantees that a neural network preserves its intended outputs under well-characterized and bounded perturbations to its inputs. Because traditional testing is often insufficient to guarantee the intended behavior of neural networks, formal methods act as a *means of compliance* to bridge this certification gap.

Formal methods have proven effective on low-dimensional tasks. Scaling these methods for high-dimensional tasks requires a major technological leap. The scope of this thesis is to develop formal verification capabilities for high-dimensional inputs, specifically pictures. Processing pictures poses a high-dimensional challenge, as each pixel acts as a distinct input, and a single picture consists of thousands of pixels. Much research on the formal robustness of pictures focuses on white-noise perturbations (l_p-norm) Mangal et al. [2023]. While some existing methods tackle more complex perturbations, they often struggle to scale to larger pictures and neural networks, or they require specific training. Additionally, these methods focus on the classification task.

Verifying robustness for these computer vision neural networks against complex perturbations raises the following questions:

[1]While aeronautical standards typically use the term "stability," we use "robustness" to align with the Machine Learning community.

- First, how do we verify robustness against complex perturbations on pictures **efficiently**?
- Second, can we extend the field to other neural network tasks such as object detection?

To answer these questions, we focused on the vision-based landing use case, specifically runway detection using the LARD dataset. As shown in the EASA roadmap (Figure 1.6, Page 17), perception systems for landing are highly complex and highly critical. By selecting this specific object detection task, we ensure that our formal verification tools push the boundaries of current research. This scenario is highly relevant for several reasons. In such scenarios, neural networks must process pictures that are subject to various environmental and sensor-based perturbations. Changes in weather or time of day alter the brightness and contrast of the input, while electronic sensors may add white-noise. Flight motions and aircraft vibrations shake the camera, causing the picture to rotate, translate, or shift. Geometric perturbations are particularly complex to verify, as they move pixel locations and introduce highly non-linear mathematical variations.

Furthermore, runway detection relies on object detection. This represents a more advanced challenge than classification. Bringing formal verification to this specific task pushes the boundaries of current research and addresses an industry need.

The next section briefly reviews the formal verification workflow previously introduced in the Background Chapter 2 (Page 26). We show which specific parts of the workflow are addressed by the contributions of this PhD.

6.2 Robustness of object detection neural network against perturbations

Robustness verification aims to verify that the output of the neural network f remains correct under perturbations to a reference picture x. This analysis generally considers three components, as illustrated in Figure 6.1:

- **Input domain definition** ($\mathcal{C}$). The perturbations around x are represented by a set $\mathcal{C}$. It encompasses all possible pictures generated by the perturbations.
- **Approximation of the neural network** ($f^a(\mathcal{C})$). $f^a(\mathcal{C})$ represents an over-approximation of the neural network output over the domain $\mathcal{C}$ using relaxations on each layer.
- **The specification set** (Y). The set of correct outputs is defined as Y. It defines the intended behavior.

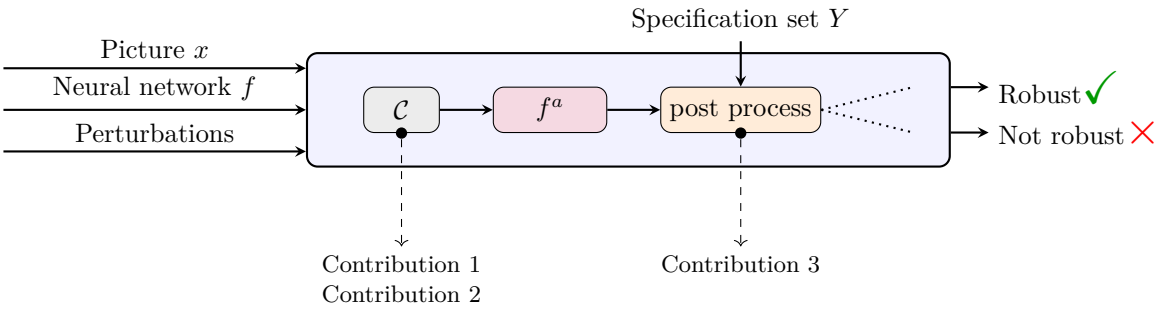


Figure 6.1: Robustness verification workflow

The goal is to prove that the neural network output is contained within the specification set Y by showing that its over-approximation $f^a(\mathcal{C})$ satisfies this condition (Property 2.3, Page 30):

$$f^a(\mathcal{C}) \subseteq Y \tag{6.1}$$

Checking this condition allows us to determine if the neural network is robust against the perturbations. If the verification result is unknown or violates the property, we consider the network is unverified for robustness.

The questions of the previous subsection (Subsection 6.1) translate to:

- **Input domain** How can we define and formalize the input domain $\mathcal{C}$ for complex perturbations efficiently? How can we verify the neural network for this input domain efficiently?
- **Robustness check** How should we define the specification set Y for object detection? Furthermore, given Y and $f^a(\mathcal{C})$, how do we check Equation 6.1?

Computing a tight over-approximation for the input domain $\mathcal{C}$ representing all possible geometrically transformed pictures is computationally demanding or often leads to loose bounds. Second, defining the intended behavior for object detection is much more complex than for classification. As shown in Figure 6.1 (dashed arrows), our research solves these challenges:

- We focused on efficiently bounding the perturbed input domain $\mathcal{C}$ for geometric transformations applied to a picture x. These transformations are highly non-linear and non-convex. This effort resulted in Contribution 1 and Contribution 2.
- Given $f^a(\mathcal{C})$ and Y, we introduce a post-processing function, based on the common metric for object detection Intersection over Union (IoU) metric, to verify robustness. This work resulted in Contribution 3.

6.3 Contributions

6.3.1 Verifying geometric robustness of neural networks

First, in our contribution 1, we provided a survey and comparison of the state-of-the-art in robustness against geometric perturbations (Chapter 3). We analyzed and compared tools such as PWL, DeepG, FGV, and Semantify-NN. There are two main directions:

- **The operator-wise bounding strategy (FGV, Semantify-NN):** This approach decomposes the geometric transformation into mathematical operators (like addition, multiplication, cosine etc) and bounds each one individually using Interval Bound Propagation (IBP) and then compose them. This strategy is fast and highly parallelizable on GPUs. However, treating each operation independently accumulates over-approximation, resulting in loose bounds. To circumvent this issue, methods partition the range of perturbations into small interval (with small interval size (IS), see Subsection 3.2.2, Page 64).
- **The Lipschitz optimization strategy (DeepG, PWL):** This approach treats the geometric transformation as a global function over the range of perturbations, in this case using its Lipschitz continuity. To create a valid over-approximation of the perturbed picture, this strategy follows three specific steps:

- **Step 1: Construction of an unsound linear bound.** An initial candidate linear bound is calculated based on a finite set of sample points within the perturbation domain. This bound is unsound as it is only guaranteed to hold for the discrete sampled points, rather than the entire continuous range.
- **Step 2: Computation of the Lipschitz constant.** A Lipschitz constant is determined for the pixel transformation over the range of perturbations. This constant defines the maximum rate at which the pixel value changes relative to the transformation parameters.
- **Step 3: Correction to a sound linear bound.** A correction term is applied to the unsound bound to ensure it holds for the range of perturbations. This correction term is found by partitioning the domain in sub-intervals and applying the Lipschitz continuity on them, creating a sound over-approximation.

DeepG and PWL used a Linear Programming (Gurobi) solver for Step 1. They use an iterative algorithm for Step 3: they rely on a priority queue that adds new sample points one by one where the over-approximation is biggest. Their formulation is hard to parallelize on GPU.

By clarifying these strategies, we identified an objective: the field needed a method that combined the tight precision of the Lipschitz optimization strategy with the GPU-accelerated speed of the operator-wise strategy.

6.3.2 The Super-DeepG approach

In our contribution 2, to resolve the computational bottlenecks of previous Lipschitz-based methods, the Super-DeepG approach (Chapter 4) reformulates the two steps:

- **Step 1: Analytical construction of the unsound bound.** An experimental observation of the Super-DeepG approach is that a small number of sample points is sufficient to define the unsound bound, as the geometric transformation remains nearly linear when evaluated over small interval sizes. The Super-DeepG approach replaces the linear programming solver with an analytical solution. A mathematical property is used: any optimal linear bound necessarily intersects $d+1$ affinely independent sample points, where d is the dimension of the transformation parameters. By enumerating a finite set of candidate sub-problems, the optimal coefficients are calculated through linear algebra. This reformulation allows the GPU parallelization across pixels and sub-intervals.
- **Step 3: Non-iterative sound correction.** The Super-DeepG approach replaces the iterative approach with a grid-based strategy. The multidimensional perturbation domain is divided into a uniform grid of N^d sub-intervals. The correction term is computed by evaluating the residuals at the center points of this grid in a single step. This grid-based method provides the same soundness and precision as the previous approaches but eliminates the need for sequential queues.

These changes allow parallel tensor operations on the GPU, resulting in a significant increase in speed. This approach maintains the high precision of the DeepG Lipschitz optimization while achieving more scalability.

6.3.3 Robustness of Object Detection to Perturbations

Third, we extended robustness verification to the object detection problem (Chapter 5). Moving beyond standard classification requires us to evaluate bounding box predictions. To measure the correctness of a predicted bounding box against a ground truth bounding

box, a common metric is the Intersection over Union (IoU). We proposed the Unimodal-IoU method to verify these bounding box predictions against perturbations such as white-noise, brightness, and contrast changes.

Applying relaxation techniques (like CROWN or IBP) to the neural network yields an over-approximation of the output, providing interval bounds for each of the four coordinates of the predicted bounding box. Finding the guaranteed minimum IoU score within this infinite set of possible bounding boxes is not straightforward. A concurrent approach Raviv et al. [2024] decomposes the IoU metric into mathematical operators (like addition, multiplication, etc) and bounds each one individually using linear relaxation and then compose them. This introduced significant over-approximation, making it difficult to prove robustness.

In our case, we analyzed the mathematical properties of the IoU function itself. We proved that the IoU metric is unimodal with respect to each coordinate independently. This unimodality property guarantees that the minimum IoU score can only occur at the extreme boundaries of the input intervals. Thanks to this property, we can find the guaranteed minimum score by evaluating the 16 corner combinations (vertices) formed by the coordinate intervals. Unimodal-IoU method provides the tightest possible verification for the given interval coordinate bounds.

6.4 Insights

Based on the methods and experiments detailed in this thesis, two key insights stand out:

- **The interval size of the range of perturbations is a key hyperparameter:** Using an incomplete method, the larger the range of perturbations we apply to a picture, the larger the resulting over-approximation $f^a(\mathcal{C})$ becomes. When our bounds become too loose, we fail to prove robustness and end up with an *Unknown* verification result. To mitigate this issue, we must partition the initial perturbation range into very small sub-intervals. Evaluating thousands of small sub-intervals is computationally heavy. Therefore, a solution is to design mathematical formulations that can be encoded into parallelizable tensor operations for GPUs.

- **There is a trade-off between complexity and precision:** Robustness verification for neural networks against complex perturbations depends on managing the trade-off between complexity and precision. Research must continue to advance in three directions.First, developing imprecise method and highly parallelizable approaches ensures that GPUs can compute bounds rapidly and scale. Second, crafting highly precise method prevents the accumulation of over-approximation that lead to *unknown* results. Last, exploring hybrid techniques that merge fast tensor operations with precise method represents a path to verify robustness against complex perturbations efficiently.

- **Treating functions as a whole outperforms operator-wise decomposition:** Decomposing a function accumulates over-approximation at every step. This is highlighted in our work in two distinct areas. For geometric transformations, the Lipschitz optimization strategy provides tighter bounds than the operator-wise bounding strategy. For object detection, the Unimodal-IoU approach outperforms methods that decompose the metric into individual operations like addition, and minimums.

6.5 Perspectives

Building on the results of this thesis, several promising directions emerge for future research.

6.5.1 Combining geometric perturbations and object detection

Currently, we treat geometric perturbations and object detection verification as separate problems. A highly relevant next step is to combine the Super-DeepG and Unimodal-IoU approaches to evaluate an object detection neural network against geometric perturbations. One difficulty is that our current method assumes a fixed ground truth. When the input picture rotates or translates, the ground truth bounding box also moves, as illustrated in Example 18. Future research must address this challenge by performing the verification with a dynamic ground truth box that adjusts its coordinates during the transformation.

Example 18 (Moving ground truth with a rotation)**.** *When rotated by an angle θ about the origin, the ground truth coordinates $[z_0^{gt}, z_1^{gt}, z_2^{gt}, z_3^{gt}]$ illustrated in Figure 6.2 (left) transform according to the rotation matrix:*

$$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

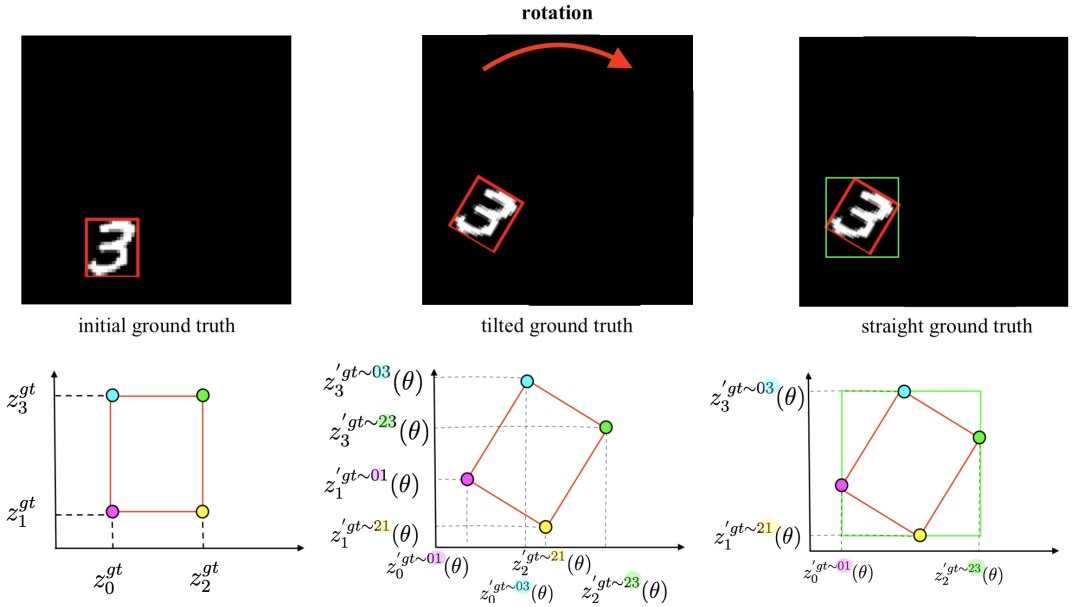


Figure 6.2: Moving ground truth

The rotated corners points $(z_0^{'gt}, z_1^{'gt})$ (bottom-left), $(z_2^{'gt}, z_3^{'gt})$ (top-right), $(z_0^{'gt}, z_3^{'gt})$ (bottom-left), $(z_2^{'gt}, z_1^{'gt})$ (bottom-right), $(z_0^{'gt}, z_3^{'gt})$ (top-left) are illustrated in Figure 6.2 (middle) and given by:

$$z_i^{'gt\sim ij}(\theta) = z_i^{gt}\cos\theta - z_j^{gt}\sin\theta,$$
$$z_j^{'gt\sim ij}(\theta) = z_i^{gt}\sin\theta + z_j^{gt}\cos\theta,$$

The indices i and j represent the ground truth coordinates forming that corner. For example, rotating the bottom-left corner (z_0^{gt}, z_1^{gt}) uses the notation ~ 01.

To ensure the rectangle remains straight and does not tilt, we must apply an adjustment so that the final coordinates, illustrated in Figure 6.2 (right), are given by:

$$
\begin{aligned}
z_0'^{gt}(\theta) &= \min\left(z_0'^{gt\sim 01}(\theta), z_0'^{gt\sim 03}(\theta), z_2'^{gt\sim 21}(\theta), z_2'^{gt\sim 23}(\theta)\right), \\
z_1'^{gt}(\theta) &= \min\left(z_1'^{gt\sim 01}(\theta), z_1'^{gt\sim 21}(\theta), z_3'^{gt\sim 03}(\theta), z_3'^{gt\sim 23}(\theta)\right), \\
z_2'^{gt}(\theta) &= \max\left(z_0'^{gt\sim 01}(\theta), z_0'^{gt\sim 03}(\theta), z_2'^{gt\sim 21}(\theta), z_2'^{gt\sim 23}(\theta)\right), \\
z_3'^{gt}(\theta) &= \max\left(z_1'^{gt\sim 01}(\theta), z_1'^{gt\sim 21}(\theta), z_3'^{gt\sim 03}(\theta), z_3'^{gt\sim 23}(\theta)\right).
\end{aligned}
$$

Furthermore, real-world systems rely on high-performance neural networks such as YOLO. Indeed, future research to combine both approaches needs to use the new methods proposed by Brückner et al. [2026].

6.5.2 Open challenges for real-world perturbations

While this thesis successfully verifies specific picture perturbations, a major constraint remains for the field: mathematically modeling the complete set of perturbations that can occur in the real world. The formal guarantees we establish apply locally. Our bounds are proven valid only around a specific, known reference picture. But real-world perturbations are incredibly complex. This complexity is not just due to the large size of the pictures or the depth of the neural networks, but also the number of possible perturbations. Even if an airport tarmac is a highly *regulated* environment, with a well-defined Operational Design Domain (ODD) Cappi et al. [2024], every single pixel in the camera's view can still vary independently. This creates a massive combinatorial space of possible pictures. Finding ways to mathematically bound and verify this enormous set of real-world an open challenge for the future of certified neural networks. During a landing approach, plausible perturbations correspond to a specific aircraft trajectory and attitude envelope. This envelope defines the approach cone. Future work can first focus on combining geometric perturbations that are physically possible within this cone. Current geometric robustness verification focuses on well known transformations but do not include projective transformations. These transformations create perspective effects where objects appear smaller as they move further away, as shown in Figure 6.3. An extension can be to deal with them.

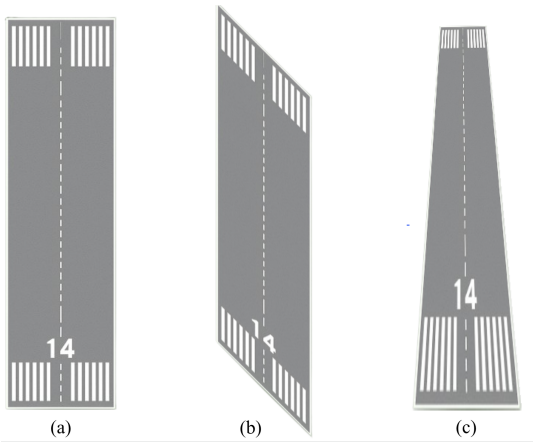


Figure 6.3: Views of a runway under geometric transformations. (a): original picture. (b) **Affine**: the runway is sheared; parallel lines are preserved but angles and aspect ratios are distorted. (c) **Projective**: the runway converges towards a vanishing point, so the far end appears narrower than the near end, mimicking perspective.

6.5.3 Adapting to alternative overlap metrics

Several extensions of $\text{IoU}_{b_{gt}}$ have been proposed in the literature to address its limitations (see Example 19). Notable examples include Generalized IoU (GIoU) Rezatofighi et al. [2019], Distance-IoU (DIoU) and Complete IoU (CIoU) Zheng et al. [2020], Wise-IoU (WIoU) Tong et al. [2023], and SIoU Gevorgyan [2022]. Investigating how the Unimodal-IoU methodology adapts to these advanced metrics represents a direction of research for formal verification for object detection systems.

Example 19 (GIoU metric - Solving IoU issue)**.** *As seen in Figure 5.2 (right) (Page 124), if the bounding boxes do not overlap, $IoU_{b_{gt}}(b) = 0$. This metric does not distinguish whether the prediction is just slightly disjoint from the ground truth or located at the opposite end of the picture, as both yield an identical score of zero. Let b be a predicted bounding box, b_{gt} be the ground truth bounding box, and b_c be their smallest enclosing bounding box. The GIoU is a function mapping $(\mathbb{R}^4)^2 \longmapsto [-1, 1]$ such that:*

$$GIoU_{b_{gt}}(b, b_c) = IoU_{b_{gt}}(b) - \frac{a(b_c) - a(b \cup b_{gt})}{a(b_c)}$$

As the distance between b and b_{gt} increases, $GIoU_{b_{gt}}(b, b_c)$ becomes increasingly negative, approaching -1.

6.6 Concluding remarks

In summary, this PhD takes a step toward the robustness verification of neural networks for safety-critical avionics. By focusing on the high-dimensional challenge of vision-based landing, we demonstrated that formal methods can successfully scale to complex computer vision tasks. Our two main approaches, the Super-DeepG and the Unimodal-IoU methods, show that reformulating mathematical problems for GPU tensorization and leveraging specific metric properties can overcome the limits of traditional verification. While formally verifying a neural network against the massive combinatorial space of real-world perturbations remains an open challenge, this work provides a small building block toward a fast, precise, and scalable verification process for the future of ML-based aeronautical safety.

Appendix A

Supplementary Materials

A.1 Deepg algorithm outline step 3

Algorithm Outline.

1) Initialization
We consider the iteration $i = 0$ that begin by considering the full domain as a single interval $[\underline{\kappa}, \overline{\kappa}]$.

- Let $\kappa_{mid} = \frac{\underline{\kappa}+\overline{\kappa}}{2}$ be the midpoint. We evaluate the residual at this center $\overline{r}\Big(\kappa_{mid}; \overline{a}, \overline{b}\Big)$.
- We initialize our current estimate of the global minimum: $\text{bestMin} = \overline{r}\Big(\kappa_{mid}; \overline{a}, \overline{b}\Big)$.
- We compute the *Lipschitz-based correction term* for this interval using Equation (4.29):

$$\overline{c} \;=\; \overline{r}\Big(\kappa_{mid}; \overline{a}, \overline{b}\Big) \;-\; \mathcal{L}_{i,j} \left|\left| \tfrac{\overline{\kappa}-\underline{\kappa}}{2} \right|\right|$$

This single interval $[\underline{\kappa}, \overline{\kappa}]$ is inserted into a priority queue q, ordered by $\overline{c}$ (the smallest $\overline{c}$ is given highest priority).

2) Iterations

1. **while** the smallest correction term in the queue $\Big(\min \overline{c}\Big)$ is less than $\text{bestMin} - \epsilon$, do:
   - *Pop* from q the interval with the smallest $\overline{c}$. Let us denote this interval as $[\kappa_a, \kappa_b]$.
   - Calculate the midpoint $\kappa_{new} = \frac{\kappa_a+\kappa_b}{2}$.
   - Subdivide $[\kappa_a, \kappa_b]$ into two sub-intervals: $[\kappa_a, \kappa_{new}]$ and $[\kappa_{new}, \kappa_b]$.
   - *For each* of these two sub-intervals (denoted generically as $[\kappa_{start}, \kappa_{end}]$):
     - Compute its midpoint $\kappa_{mid} = \frac{\kappa_{start}+\kappa_{end}}{2}$.
     - Evaluate $\overline{r}\Big(\kappa_{mid}; \overline{a}, \overline{b}\Big)$.
     - Update the global minimum estimate: $\text{bestMin} = \min\Big(\text{bestMin},\ \overline{r}\Big(\kappa_{mid}; \overline{a}, \overline{b}\Big)\Big)$.
     - Compute the correction term for this sub-interval: $\overline{c} = \overline{r}\Big(\kappa_{mid}; \overline{a}, \overline{b}\Big) - \mathcal{L}_{i,j} \left|\left| \tfrac{\kappa_{end}-\kappa_{start}}{2} \right|\right|$.
     - If $\overline{c} < \text{bestMin} - \epsilon$, *push* this new sub-interval into q with priority $\overline{c}$.
2. **end while**

Bibliography

Acubed. Wayfinder project, 2019. URL https://acubed.airbus.com/blog/wayfinder/how-wayfinder-is-using-neural-networks-for-vision-based-autonomous-landing/.

Airbus and Onera. Attol project, 2020. URL https://www.airbus.com/en/newsroom/press-releases/2020-06-airbus-concludes-attol-with-fully-autonomous-flight-tests.

Kahn Mason Ashish Goel, Benjamin Van Roy. Introduction to Optimization. https://web.stanford.edu/~ashishg/msande111/notes/chapter3.pdf, 2008.

Giovanni Balduzzi, Martino Ferrari Bravo, Anna Chernova, Calin Cruceru, Luuk van Dijk, Peter de Lange, Juan Jerez, Nathanaël Koehler, Mathias Koerner, Corentin Perret-Gentil, et al. Neural network based runway landing guidance for general aviation autoland. Technical report, United States. Department of Transportation. Federal Aviation Administration ..., 2021.

Mislav Balunovic, Maximilian Baader, Gagandeep Singh, Timon Gehr, and Martin Vechev. Certifying geometric robustness of neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

Ben Batten, Yang Zheng, Alessandro De Palma, Panagiotis Kouvaros, and Alessio Lomuscio. Verification of geometric robustness of neural networks via piecewise linear approximation and lipschitz optimisation. In *ECAI 2024*, pages 2362–2369. IOS Press, 2024.

Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization*, volume 6. Athena scientific Belmont, MA, 1997.

Siddhartha Bhattacharyya, Darren Cofer, David Musliner, Joseph Mueller, and Eric Engstrom. Certification considerations for adaptive systems. In *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 270–279. IEEE, 2015.

Patrick Billingsley. *Probability and Measure*. John Wiley & Sons, 3rd edition, 1995. Discusses the integration of continuous probability measures over sets of Lebesgue measure zero.

Boeing. Beacon project, 2025. URL https://www.eurocontrol.int/sites/default/files/2025-04/20250422-flyai-forum-session1-jahn-boeing.pdf.

Frédéric Boniol, Adrien Chan-Hon-Tong, Alexandre Eudes, Stéphane Herbin, Guy Le Besnerais, Claire Pagetti, and Martial Sanfourche. Challenges in the certification of computer vision-based systems. *Aerospace Lab*, 2020.

Elena Botoeva, Panagiotis Kouvaros, Jan Kronqvist, Alessio Lomuscio, and Ruth Misener. Efficient verification of relu-based neural networks via dependency analysis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3291–3299, 2020.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Christopher Brix, Stanley Bak, Taylor T Johnson, and Haoze Wu. The fifth international verification of neural networks competition (vnn-comp 2024): Summary and results. *arXiv preprint arXiv:2412.19985*, 2024.

Benedikt Brückner and Alessio Lomuscio. Verification of neural networks against convolutional perturbations via parameterised kernels. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 27215–27223, 2025.

Benedikt Brückner, Alejandro Mercado, Yanghao Zhang, Panagiotis Kouvaros, and Alessio Lomuscio. Ioucert: Robustness verification for anchor-based object detectors. *arXiv preprint arXiv:2603.03043*, 2026.

Simon Burton, Iwo Kurzidem, Adrian Schwaiger, Philipp Schleiss, Michael Unterreiner, Torben Graeber, and Philipp Becker. Safety assurance of machine learning for chassis control functions. In *International Conference on Computer Safety, Reliability, and Security*, pages 149–162. Springer, 2021.

Cyril Cappi, Noémie Cohen, Mélanie Ducoffe, Christophe Gabreau, Laurent Gardes, Adrien Gauffriau, Jean-Brice Ginestet, Franck Mamalet, Vincent Mussot, Claire Pagetti, et al. How to design a dataset compliant with an ml-based system odd? *arXiv preprint arXiv:2406.14027*, 2024.

Arthur Clavière, Eric Asselin, Christophe Garion, and Claire Pagetti. Vérification formelle de la sûreté d'un système contrôlé par réseaux de neurones. *ACTES DES 20ÈMES JOURNÉES*, page 5, 2021.

Arthur Clavière, Laura Altieri Sambartolomé, Eric Asselin, Christophe Garion, and Claire Pagetti. Verification of machine learning based cyber-physical systems: a comparative study. In *Proceedings of the 25th ACM International Conference on Hybrid Systems: Computation and Control*, pages 1–16, 2022.

Arthur Clavière, Dmitrii Kirov, and Darren Cofer. How to verify generalization capability of a neural network with formal methods. In *International Symposium on AI Verification*, pages 136–155. Springer, 2025.

Noémie Cohen, Mélanie Ducoffe, Ryma Boumazouza, Christophe Gabrea, Claire Pagetti, Xavier Pucel, and Audrey Galametz. Verifiou–robustness of object detection to perturbations. In *2025 AIAA DATC/IEEE 44th Digital Avionics Systems Conference (DASC)*, pages 1–10. IEEE, 2025.

Noemie Cohen, Mélanie Ducoffe, Christophe Gabreau, Claire Pagetti, and Xavier Pucel. Certified geometric robustness - super-deepg. In *Proceedings of the 29th ACM International Conference on Hybrid Systems: Computation and Control*, 2026a.

Noemie Cohen, Christophe Gabreau, Mélanie Ducoffe, Claire Pagetti, and Xavier Pucel. Towards the use of formal methods in the certification of vision-based landing systems using high-dimensional ml algorithms. In *13th European Congress of Embedded Real Time Systems (ERTS)*, 2026b.

Ronan Collobert, Samy Bengio, and Johnny Mariéthoz. Torch: a modular machine learning software library. Technical Report IDIAP-RR 02-46, Idiap, 2002.

Francesco Croce, Maksym Andriushchenko, Vikash Sehwag, Edoardo Debenedetti, Nicolas Flammarion, Mung Chiang, Prateek Mittal, and Matthias Hein. Robustbench: a standardized adversarial robustness benchmark. *arXiv preprint arXiv:2010.09670*, 2020.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Daedalean and EASA. Concepts of design assurance for neural networks (codann) ii. *European Union Aviation Safety Agency, Tech. Rep*, 1, 2021.

Mathieu Damour, Florence De Grancey, Christophe Gabreau, Adrien Gauffriau, Jean-Brice Ginestet, Alexandre Hervieu, Thomas Huraux, Claire Pagetti, Ludovic Ponsolle, and Arthur Clavière. Towards certification of a reduced footprint acas-xu system: A hybrid ml-based solution. In *International Conference on Computer Safety, Reliability, and Security*, pages 34–48. Springer, 2021.

Hervé Delseny, Christophe Gabreau, Adrien Gauffriau, Bernard Beaudouin, Ludovic Ponsolle, Lucian Alecu, Hugues Bonnin, Brice Beltran, Didier Duchel, Jean-Brice Ginestet, et al. White paper machine learning in certified systems. *arXiv preprint arXiv:2103.10529*, 2021.

Jb Dennis. First version of data flow procedure language. *Lecture Notes in Computer Science*, 19:362–376, 1974.

Melanie Ducoffe. Decomon: Automatic certified perturbation analysis of neural networks, 2022. 2023-11-20.

Mélanie Ducoffe, Maxime Carrere, Léo Féliers, Adrien Gauffriau, Vincent Mussot, Claire Pagetti, and Thierry Sammour. Lard–landing approach runway detection–dataset for vision based landing. *arXiv preprint arXiv:2304.09938*, 2023.

EASA. Software aspects of certification. Certification Memorandum EASA CM-SWCEH-002, Issue 01, Revision 01, European Aviation Safety Agency (EASA), March 2012. URL https://www.easa.europa.eu/sites/default/files/dfu/certification-docs-certification-memorandum-EASA-CM-SWCEH-002-Issue-01-Rev-01-Software-Aspects-of-Certification.pdf.

EASA. Certification specifications for large aeroplanes, 2018.

EASA. Easa concept paper: First usable guidance for level 1 and 2 machine learning applications, 2023.

Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *International symposium on automated technology for verification and analysis*, pages 269–286. Springer, 2017.

Yizhak Elboher, Raya Elsaleh, Omri Isac, Mélanie Ducoffe, Audrey Galametz, Guillaume Povéda, Ryma Boumazouza, Noémie Cohen, and Guy Katz. Robustness assessment of a runway object classifier for safe aircraft taxiing. In *2024 AIAA DATC/IEEE 43rd Digital Avionics Systems Conference (DASC)*, pages 1–6, 2024a. doi: 10.1109/DASC62030.2024.10748680.

Yizhak Yisrael Elboher, Raya Elsaleh, Omri Isac, Mélanie Ducoffe, Audrey Galametz, Guillaume Povéda, Ryma Boumazouza, Noémie Cohen, and Guy Katz. Robustness assessment of a runway object classifier for safe aircraft taxiing. In *2024 AIAA/IEEE 43rd Digital Avionics Systems Conference (DASC)*, 2024b. doi: 10.1109/DASC62030.2024.10748680.

Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

Christophe Gabreau, Nicolas Valot, Filipo Perotto, and Claire Pagetti. Implementation and certification of ml-based surrogate models in avionic systems. In *13th European Congress of Embedded Real Time Systems (ERTS)*, 2026.

Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2018.

Zhora Gevorgyan. Siou loss: More powerful learning for bounding box regression. *arXiv preprint arXiv:2205.12740*, 2022.

Antoine Girard. Reachability of uncertain linear systems using zonotopes. In *International workshop on hybrid systems: Computation and control*, pages 291–305. Springer, 2005.

Xingwu Guo, Ziwei Zhou, Yueling Zhang, Guy Katz, and Min Zhang. Occrob: efficient smt-based occlusion robustness verification of deep neural networks. In *International conference on tools and algorithms for the construction and analysis of systems*, pages 208–226. Springer, 2023.

Harleen Hanspal, Alessandro De Palma, and Alessio Lomuscio. Robustness to perturbations in the frequency domain: Neural network verification and certified training. In *Proceedings of the Winter Conference on Applications of Computer Vision (WACV) Workshops*, pages 628–637, 2025.

Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

Richard Hawkins, Colin Paterson, Chiara Picardi, Yan Jia, Radu Calinescu, and Ibrahim Habli. Guidance on the assurance of machine learning in autonomous systems (amlas). *arXiv preprint arXiv:2102.01564*, 2021.

Juha Heinonen. *Lectures on analysis on metric spaces*. Springer Science & Business Media, 2001.

Patrick Henriksen, Kerstin Hammernik, Daniel Rueckert, and Alessio Lomuscio. Bias field robustness verification of large neural image classifiers. In *BMVC*, page 202, 2021.

Frederick S Hillier and Gerald J Lieberman. *Introduction to operations research*. McGraw-Hill, 2015.

Eric Jenn, Alexandre Albore, Franck Mamalet, Grégory Flandin, Christophe Gabreau, Hervé Delseny, Adrien Gauffriau, Hugues Bonnin, Lucian Alecu, Jérémy Pirard, et al. Identifying challenges to the certification of machine learning for safety critical systems. In *European congress on embedded real time systems (ERTS 2020)*, volume 1, 2020.

Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International conference on computer aided verification*, pages 97–117. Springer, 2017.

Matthias König, Annelot W. Bosman, Holger H. Hoos, and Jan N. van Rijn. Critically assessing the state of the art in neural network verification. *Journal of Machine Learning Research*, 25:1–53, 2024.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 2002.

Zongxin Liu, Lijia Yu, Tao Lin, Zhiming Chi, and Lijun Zhang. Certifying the full yolo pipeline: A probabilistic verification approach. In *The Fourteenth International Conference on Learning Representations*, 2026.

Wei Ma and Jun Lu. An equivalence of fully connected layer and convolutional layer. *arXiv preprint arXiv:1712.01252*, 2017.

Ravi Mangal, Klas Leino, Zifan Wang, Kai Hu, Weicheng Yu, Corina Pasareanu, Anupam Datta, and Matt Fredrikson. Is certifying lp robustness still worthwhile? *arXiv preprint arXiv:2310.09361*, 2023.

Jeet Mohapatra, Tsui-Wei Weng, Pin-Yu Chen, Sijia Liu, and Luca Daniel. Towards verifying robustness of neural networks against a family of semantic perturbations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 244–252, 2020.

Christoph Müller, François Serre, Gagandeep Singh, Markus Püschel, and Martin Vechev. Scaling polyhedral neural network verification on gpus. *Proceedings of Machine Learning and Systems*, 3:733–746, 2021.

Ashutosh Kumar Nirala and Soumalya Sarkar. Towards certified object detectors: Certified runway detection using yolo. In *2025 IEEE International Conference on Image Processing (ICIP)*, pages 827–832. IEEE, 2025.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Amir Pnueli. The temporal logic of programs. In *18th annual symposium on foundations of computer science (sfcs 1977)*, pages 46–57. ieee, 1977.

Avraham Raviv, Yizhak Y Elboher, Michelle Aluf-Medina, Yael Leibovich Weiss, Omer Cohen, Roy Assa, Guy Katz, and Hillel Kugler. Formal verification of object detection. *arXiv preprint arXiv:2407.01295*, 2024.

Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 658–666, 2019.

RTCA/EUROCAE. DO-254/ED-80 - Design Assurance Guidance For Airborne Electronic Hardware, 2000.

RTCA/EUROCAE. Ed-12/do-178c “software considerations in airborne systems and equipment certification", 2011.

RTCA/EUROCAE. DO-254/ED-216 - formal methods supplement to do-178c and do-278a, 2011.

RTCA/EUROCAE. AMC/AC-20-115d - airborne software development assurance, 2017.

RTCA/EUROCAE. AMC/AC-20-152a - development assurance for airborne electronic hardware, 2020.

W Rudin. Principles of mathematical analysis. *3rd ed.*, 1976.

SAE. Aerospace Recommended Practices ARP5151- Safety Assessment of General Aviation Airplanes and Rotorcraft in Commercial Service, 2013.

SAE. Aerospace Recommended Practices ARP5150- Safety Assessment of Aircraft in Commercial Service, 2019.

SAE/EUROCAE. Aerospace Recommended Practices ARP4754a/ED-79A- Development of Civil Aircraft and Systems, 2010.

SAE/EUROCAE. Aerospace Recommended Practices ARP4761A/ED-135- guidelines and methods for conducting the safety assessment process on civil airborne systems and equipement, 2020.

Simone Scardapane. Alice's adventures in a differentiable wonderland–volume i, a tour of the land. *arXiv preprint arXiv:2404.17625*, 2024.

Zhouxing Shi, Qirui Jin, Zico Kolter, Suman Jana, Cho-Jui Hsieh, and Huan Zhang. Neural network verification with branch-and-bound for general nonlinearities. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 315–335. Springer, 2025.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–30, 2019.

Jean Souyris, Virginie Wiels, David Delmas, and Hervé Delseny. Formal verification of avionics software products. In *International symposium on formal methods*, pages 532–546. Springer, 2009.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Vincent Tjeng, Kai Yuanqing Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations*, 2019.

Zanjia Tong, Yuhang Chen, Zewei Xu, and Rong Yu. Wise-iou: Bounding box regression loss with dynamic focusing mechanism. *arXiv preprint arXiv:2301.10051*, 2023.

Kush R Varshney. Engineering safety in machine learning. In *2016 Information Theory and Applications Workshop (ITA)*, pages 1–5. IEEE, 2016.

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. *Advances in neural information processing systems*, 31, 2018.

Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. *Advances in neural information processing systems*, 34:29909–29921, 2021.

Lily Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane Boning, and Inderjit Dhillon. Towards fast computation of certified robustness for relu networks. In *International Conference on Machine Learning*, pages 5276–5285. PMLR, 2018.

WG-114/SAE/EUROCAE. Air6988/er-022 artificial intelligence in aeronautical systems: statement of concerns, 2021.

Jamie P White and Hassan Reza. Deriving do-178c requirements within the appropriate level of hierarchy. In *International Conference on Software Engineering Advances*, pages 430–435, 2012.

Weiming Xiang, Hoang-Dung Tran, and Taylor T Johnson. Reachable set computation and safety verification for neural networks with relu activations. *arXiv preprint arXiv:1712.08163*, 2017.

Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. Automatic perturbation analysis for scalable certified robustness and beyond. *Advances in Neural Information Processing Systems*, 33:1129–1141, 2020a.

Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. *arXiv preprint arXiv:2011.13824*, 2020b.

Pengfei Yang, Renjue Li, Jianlin Li, Cheng-Chao Huang, Jingyi Wang, Jun Sun, Bai Xue, and Lijun Zhang. Improving neural network verification through spurious region guided refinement. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 389–408. Springer, 2021.

Rem Yang, Jacob Laurel, Sasa Misailovic, and Gagandeep Singh. Provable defense against geometric transformations. *arXiv preprint arXiv:2207.11177*, 2022.

Huan Zhang. VNN-COMP 2024 TinyImageNet Benchmark. https://github.com/huanzhang12/vnncomp2024_tinyimagenet_benchmark, 2024.

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. *Advances in neural information processing systems*, 31, 2018.

Yunruo Zhang, Tianyu Du, Shouling Ji, and Shanqing Guo. Drown: Towards tighter lirpa-based robustness certification. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 6212–6229, 2025.

Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. Distance-iou loss: Faster and better learning for bounding box regression. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 12993–13000, 2020.

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

Université de Toulouse

Titre : Vérification formelle des systèmes à haute dimension fondés sur l'apprentissage machine (ML)

Mots clés : Vérification formelle, Sûreté de fonctionnement, Solveur

Résumé : Cette thèse porte sur la vérification formelle pour garantir la robustesse de modèles de machine learning dédiés à la détection de pistes d'atterrissage. Dans un premier temps, les recherches se sont concentrées sur les perturbations plausibles liées au domaine opérationnel (ODD), menant au développement d'une méthodologie ainsi que d'un outil visant à démontrer la robustesse des modèles face aux transformations géométriques (rotation, translation, échelle, etc.). Cet outil utilise une optimisation Lipschitz et des relaxations linéaires pour encapsuler l'ensemble des images perturbées.
Par ailleurs, les garanties de robustesse, initialement appliquées à la classification, ont ensuite été étendues à la détection d'objets, un domaine jusqu'alors moins exploré. Une méthode de vérification a ainsi été mise au point en formalisant la métrique d'Intersection sur Union (Intersection over Union - IoU) pour des perturbations de luminosité, de bruit blanc et de contraste.
Enfin, les enjeux de certification (ED-324/ARP-6983, concept papers de l'EASA) appliqués au cas d'utilisation de la détection de piste, ont été définis.

Title: Formal verification of high-dimensional Machine Learning (ML) based systems

Key words: Formal verification, Safety, Solvers

Abstract: This thesis focuses on formal verification to ensure the robustness of machine learning models dedicated to runway detection. Initially, the research concentrated on plausible perturbations related to the Operational Design Domain (ODD), leading to the development of a methodology and a tool aimed at demonstrating the robustness of the models against geometric transformations (rotation, translation, scaling, etc.). This tool uses Lipschitz optimization and linear relaxations to encapsulate the entire set of perturbed images.
Furthermore, the robustness guarantees, initially applied to classification, were subsequently extended to object detection, a domain that had previously been less explored. A verification method was thus developed by formalizing the Intersection over Union (IoU) metric for brightness, white noise, and contrast perturbations.
Finally, the certification challenges (ED-324/ARP-6983, EASA concept papers) for object detection models, applied to the runway detection use case, were defined.