**UNIVERSITY OF SOUTHAMPTON**
Faculty of Engineering and Physical Sciences
School of Electronics and Computer Science

A project report submitted for the award of
B.Sc. Computer Science

Supervisor: Ekaterina Komendantskaya
Examiner: Hikmat Farhat

# Verification of a computer vision model

by Robert Moran

April 29, 2025

**Statement of Originality**

- I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.
- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

***You must change the statements in the boxes if you do not agree with them.***

We expect you to acknowledge all sources of information (e.g. ideas, algorithms, data) using citations. You must also put quotation marks around any sections of text that you have copied without paraphrasing. If any figures or tables have been taken or modified from another source, you must explain this in the caption and cite the original source.

**I have acknowledged all sources, and identified any content taken from elsewhere.**

If you have used any code (e.g. open-source code), reference designs, or similar resources that have been produced by anyone else, you must list them in the box below. In the report, you must explain what was used and how it relates to the work you have done.

**I have used PyTorch, alpha-beta-CROWN, the GTSRB dataset, NumPy, torchsummary, sci-kit learn. Some of my code is based on the PyTorch documentation, and example code given by my supervisor.**

You can consult with module teaching staff/demonstrators, but you should not show anyone else your work (this includes uploading your work to publicly-accessible repositories e.g. Github, unless expressly permitted by the module leader), or help them to do theirs. For individual assignments, we expect you to work on your own. For group assignments, we expect that you work only with your allocated group. You must get permission in writing from the module teaching staff before you seek outside assistance, e.g. a proofreading service, and declare it here.

**I did all the work myself, or with my allocated group, and have not helped anyone else.**

We expect that you have not fabricated, modified or distorted any data, evidence, references, experimental results, or other material used or presented in the report. You must clearly describe your experiments and how the results were obtained, and include all data, source code and/or designs (either in the report, or submitted as a separate file) so that your results could be reproduced.

**The material in the report is genuine, and I have included all my data/code/designs.**

We expect that you have not previously submitted any part of this work for another assessment. You must get permission in writing from the module teaching staff before re-using any of your previously submitted work for this assessment.

**I have not submitted any part of this work for another assessment.**

If your work involved research/studies (including surveys) on human participants, their cells or data, or on animals, you must have been granted ethical approval before the work was carried out, and any experiments must have followed these requirements. You must

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND PHYSICAL SCIENCES
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

A project report submitted for the award of B.Sc. Computer Science

by Robert Moran

The increasing prevalence of deep learning models throughout safety critical environments makes guaranteeing the performance of a model paramount. This project aims to create guarantees about the robustness of road sign detection using formal verification of neural networks.

# Contents

# List of Figures

# Chapter 1

# Introduction

In recent years the use of machine learning has become increasingly widespread, especially for allowing a computer to comprehend image data, commonly referred to as computer vision. A key application of computer vision is in autonomous vehicles, where it is paramount to the safety of users and nearby pedestrians that it can reliably understand the environment it is moving through.

However, it has been shown that often times state-of-the-art machine learning techniques can be extremely sensitive to very minor alterations in input data [1], causing poor performance in cases where it's not expected. Techniques such as adversarial attacks demonstrate how imperceptible differences in input data can cause models to give wrong outputs.

Because of this an important feature to consider when creating a computer vision model for a safety critical environment is robustness. Creating a model that can function in real world conditions with different lighting and image noise is key. Furthermore, it's vital you attempt to create a formal guarantee of this robustness.

Consequently, this project will focus on creating robust recognition of road signs from the GTSRB dataset [2].

## 1.1 Goals & Scope

This project aims to firstly create a computer vision model capable of accurately recognising road signs from the GTSRB [2] dataset, then use a variety of methods to improve the models robustness to various augmentations.

Next I will use verification techniques to make more formal claims about the robustness of the model. As a stretch I would like to train for and verify the groups constraint, shown in [3].

# Chapter 2

# Background and report of literature search

## 2.1 Image Recognition

Machine learning algorithms are ideal for image recognition, due to the complexity of the problem it would be rather tedious to manually code, and "almost impossible" [4] to make it accurate across a large set of data. In contrast, machine learning techniques like neural networks are much more ideal for this problem.

Traditional neural networks struggle with the high dimensionality of image data; even small images can have thousands of dimensions and larger images can be in the millions. Previously instead of feeding an image directly to a neural network, feature vectors would be derived from the image, such as the edges of shapes and their curvature [5], which could then be fed to the network. These features aimed to take only the most important properties of the image, so that the network has a smaller amount of data to deal with.

However, this gives rise to numerous problems; manually the optimal features to extract is time-consuming, and often specific to the data you're working with. This is a problem convolutional neural networks aim to solve.

### 2.1.1 Convolutional Neural Networks (CNN)

These models are composed of 3 main types of layers: convolutional, pooling and fully connected. The key part here is the convolutional layer. As its name implies

it applies a convolution to the image, using a kernel with weights learned through training [6]. What makes this ideal for images is that a CNN can take images as a structured input, where nearby pixels are inherently connected to each other by the convolution process, instead of taking a single vector as input like traditional neural networks.

This is an important characteristic for images as nearby pixels are inherently linked to each other [4], and form important features like the edges of shapes in the image. By considering groups of pixels at a time using a convolution, the knowledge of this relation is effectively baked into architecture of the network [4].

CNN's like AlexNet [7] and GoogLeNet [8] achieved first place in ILSVRC2012 and ILSVRC2014 respectively, both using the CNN architecture. The excellent performance by these models has shown that CNNs are significantly more powerful than standard neural networks in the domain of image recognition.

The TensorFlow library [9] conveniently offers a way of creating and training a CNN, as well as built in ways of resizing images and normalising the colour space, two important features required when creating an image recognition system. It also supports GPU acceleration for training, which was a key innovation required for the success of AlexNet [7].

## 2.2 Neural Network Safety Properties

### 2.2.1 What is the problem?

Neural networks are black boxes. Even small models are often composed of hundreds of thousands of weights, and this extreme complexity makes them incredibly difficult to interpret. Because of this it can be hard to know where a model might fail, but for a long time it was typically believed a training set would give an acceptable metric for how well a model generalises.

Yet, in Szegedy *et al.* [10] it was shown that "imperceptible" alterations to an image could cause a misclassification, proof that our understanding of robustness needed further studying.

### 2.2.2 Robustness as a Property

Robustness can be defined in various ways, but as a general definition a robust model would be able to generalise to cases where images are suboptimal, such as partial occlusion, bad weather, or excessive random noise [11].

A more formal definition is $\epsilon$-ball robustness. If all possible inputs contained in a ball of radius $\epsilon$ around some image $x$ are predicted to be in the same class as $x$, then a model would be considered $\epsilon$-ball robust around $x$ [12].

In the case of images, an epsilon ball around an image would contain all variations of that image where each pixel can deviate by up to $\pm\epsilon$.

However, as neural networks output a confidence score for each class, you need to consider more than just what class was predicted, but the confidence in that prediction as well.

Casadio *et al.* [12] discussed several ways of defining robustness as a constraint for the model to satisfy. For example standard robustness was defined as:

$$SR(\epsilon, \delta, \hat{x}) \triangleq \forall x : |x - \hat{x}| \leq \epsilon \Rightarrow ||f(x) - f(\hat{x})|| \leq \delta$$

FIGURE 2.1: Standard robustness (reproduced from [12]).

Where $\epsilon$ and $\delta$ are defined by the user, and $\hat{x}$ is the original image. Satisfying this definition would mean all values within the $\epsilon$-ball around $\hat{x}$ have a similar predicted value to $\hat{x}$.

By defining robustness as a constraint we can use frameworks like Vehicle [13] and DL2 [14] to combine the property with the loss function, allowing us to conveniently train a network with the property in mind. It should be noted that to use these frameworks the constraint must be differentiable, so that gradient descent can be calculated.

In their testing Casadio *et al.* [12] found that while Lipschitz robustness was by far the hardest to constraint to satisfy, training for it made other forms of robustness more likely to be met.

Further ideas of robustness were introduced by Flinkow *et al.* [3], where similar road signs were grouped together, and the model was punished for making predictions that spanned several groups, forcing the model to try and only predict one group at a time.

### 2.2.3 Training for Robustness

There are various methods of improving robustness for a model, such as preprocessing steps like normalising the brightness of an image, or using lower resolution images. However typically the best approach is to specifically train for robustness. This comes in the form of introducing altered data to the training set, but there are a variety of ways one can alter the images.

#### 2.2.3.1 Data Augmentation

One method to create altered images is data augmentation. The goal here is to create new images with deviations you could reasonably expect, such as translations, rotations, changes in brightness and random noise [15]. This is often used to prevent overfitting, such as in AlexNet [7] where translations and altered brightness were able to reduce the error on the test set. While augmentation is successful at improving performance and reducing overfitting, it doesn't necessarily improve performance when it comes to $\epsilon$-ball robustness.

#### 2.2.3.2 Adversarial Examples

An adversarial example is an altered image deliberately designed to be misclassified by a model with "high confidence" [1]. The idea is that an adversarial example should still be easily recognisable by a human, and visually almost the exact same as the original image, but despite this the model will completely misclassify it. Adversarial examples generally highlight how deep learning models don't get the fundamental concepts of the data they're working with [16], and that building robustness against this manner of attack is key to making a model act consistently and safely. Figure 2.2 is an example of how subtle the changes can be.

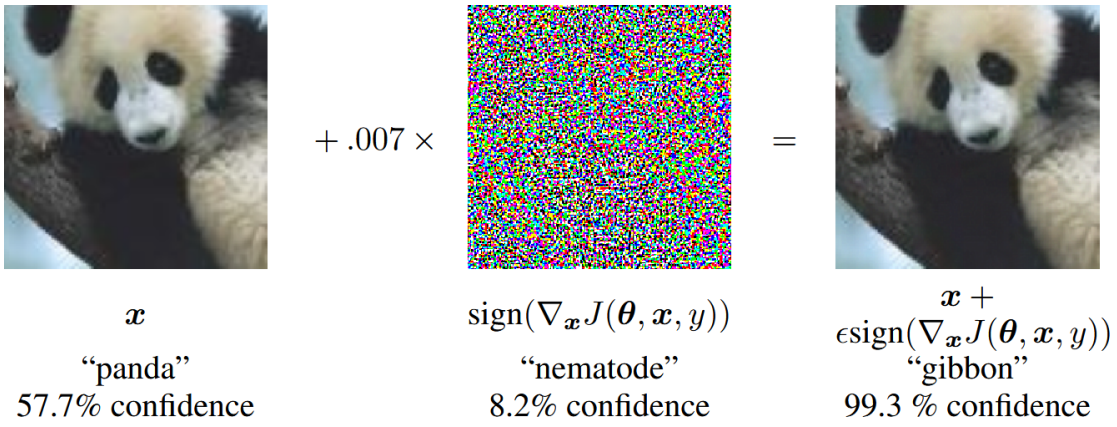



FIGURE 2.2: Example of an adversarial image (reproduced from [1]).

#### 2.2.3.3 Fast Gradient Sign Method (FGSM)

FGSM is a method of generating an adversarial example by calculating the sign of the gradient with respect to the input.

The gradient here represents which direction to move an input value to maximise the loss function; in the case of an image the input values would be the individual colour values of each pixel.

Where $x$ is the input, $L$ as the loss function, and $\epsilon$ as a parameter set by the user to determine how large the perturbations will be. The formula will be:

$$x_{\text{adv}} = x + \epsilon * \text{sign}(\nabla_x L(\theta, x, y))$$

FIGURE 2.3: FGSM equation (adapted from [1]).

However, FGSM has been found to be ineffective; due to its simple one-step nature it cannot properly explore the full space of perturbations to find an optimal adversary [16].

#### 2.2.3.4 Projected Gradient Descent (PGD)

PGD is very similar to FGSM, as its formula is almost identical. However, unlike FGSM, PGD is an iterative method. At each step we add the sign of the gradient

to the result of the previous iteration instead of the original image. This is then projected back into the $\ell_\infty$ ball around the original image. In the below equation $\Pi_x$ represents the function being used to map values outside the $\ell_\infty$ ball around the original image back inside.

$$x_{\text{adv}}^{t+1} = \Pi_x(x_{\text{adv}}^t + \epsilon * \text{sign}(\nabla_x L(\theta, x, y)))$$

FIGURE 2.4: PGD equation (adapted from [17]).

This function manages to keep the adversarial example within a specified distance from the image, while managing to achieve a better result than FGSM, at the cost of computation time.

Typically, PGD starts at a random point within the $\ell_\infty$ ball, and restarts at random intervals in order to properly explore the space and avoid local maxima. PGD can however be run without these features to save performance. [17].

Adversarial training in particular can be quite expensive computationally, due to the time taken to both generate the new images (must be done each epoch) and then train on them. In some cases it may even require a more complex model to properly model the data robustly.

## 2.3 Verification

### 2.3.1 What is verification?

Verification is the act of creating a formal guarantee about the behaviour of a model [18], in our case this property would be centred around $\epsilon$ ball robustness. This differs from simply training for robustness using techniques like PGD shown above; generating adversarial examples with PGD makes no guarantee there isn't a stronger adversary within the $\epsilon$ ball. This means even if your network is immune to adversaries generated by PGD, there is no formal guarantee of $\epsilon$ ball robustness. Verification of the neural network however will give you a definitive answer on this.

Several attempts to verify neural networks have been made, starting with work done by Pulina and Tacchella [19], but their approach was very limited in scope, only verifying very small networks with one hidden layer.

Verifying large networks is incredibly difficult, due to the complexity of verifying such a large space, as well as their non-linear nature [18]. Because of this the size and architecture of a model is a constant concern when attempting to verify, and needs to be kept small.

Regardless, later work done by Katz *et al.* [18] was able to verify much larger networks up to eight hidden layers, on the condition that the ReLU activation function is used. This detail isn't particularly problematic as ReLU is generally the preferred function for many models [20]. Further work done by Fischer *et al.* [14] has been able to introduce training with properties in mind by transforming them into a loss function and used in training.

The Verification of Neural Networks Competition (VNN Comp) [21] has been held yearly since 2020, comparing several verifiers on a set of challenges. The top scorers were $\alpha, \beta$-Crown [22]–[27]

Marabou [28] and PyRat [29]. While $\alpha, \beta$-Crown performs the best it is unable to implement constraints into the loss function.

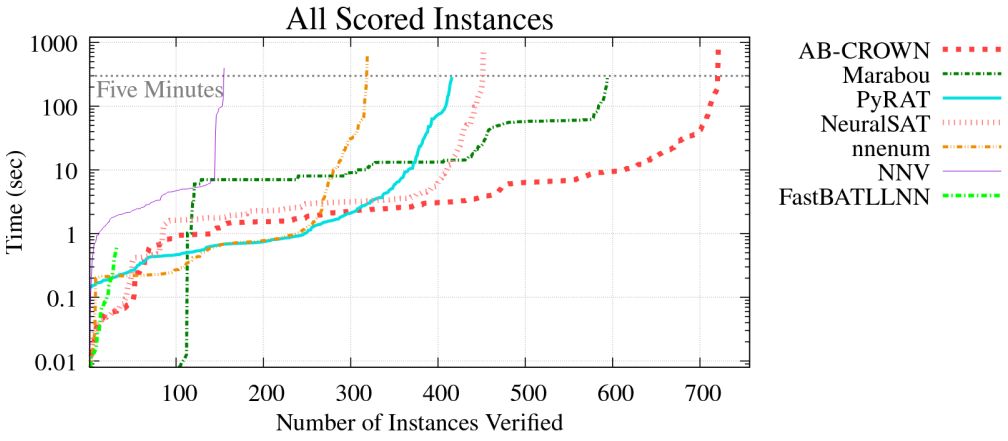



FIGURE 2.5: Scores of various verifiers from VNN Comp (Reproduced from [21])

### 2.3.2 Verification Cycle

Casadio *et al.* [12] described verification as a "continuous cycle", shown in Figure 2.6

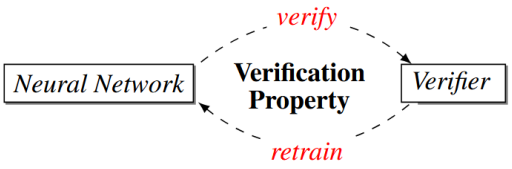



FIGURE 2.6: "Continuous Verification Cycle" (Reproduced from [12])

The cycle consists of two steps, first training the model and then using a verifier to verify it. If the model architecture is too large or has layers unsupported by the verifier, you'd expect an error, otherwise the verifier will give you the percentage of the dataset that satisfies the constraint. You can then retrain the model with the overarching goal of increasing the verification score each iteration.

# Chapter 3

# Design

## 3.1 Model

As my base model I have decided to use a Convolutional Neural Network based on LeNet-5 [4], shown in Figure 3.1. My model features a key difference in that it uses the ReLU activation function. This change serves a twofold purpose, firstly the ReLU activation function is more commonly supported amongst verifiers due to its more linear nature, and secondly ReLU boasts improved performance compared to the original tanh function. The final layer will of course also be modified to fit the 43 classes instead of the 10 used for MNIST.

Verification is a very expensive process, directly affected by the complexity of the model verified, so it was key to choose a deliberately small but competent model.

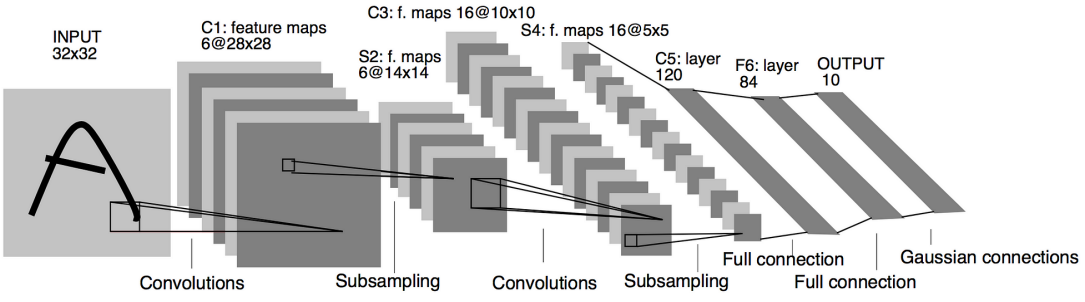



FIGURE 3.1: LeNet-5 Architecture (reproduced from [4]).

### 3.1.1 Training

The model will be trained multiple times, firstly using standard training to create a baseline performance. This will then be followed training for the property in mind, using adversarial training with Projected Gradient Descent [17].

## 3.2 Robustness Verification

For verification of robustness, I'll be using the Classification Robustness property, presented in Casadio *et al.* [12].

This property ensures the model correctly classifies every member of the epsilon ball, formularised as shown in Figure 3.2.

$$CR(\epsilon, \hat{x}) \triangleq \forall x : |x - \hat{x}| \leq \epsilon \implies \text{argmax Model}(\hat{x}) = y$$

FIGURE 3.2: Classification Robustness (adapted from [12]).

This property was chosen instead of similar properties with more desirable qualities, like Lipschitz or Standard robustness [12], because it is much simpler to implement, requiring only one call to the model. This means many verifiers can implement and verify the property.

## 3.3 Groups Constraint Verification

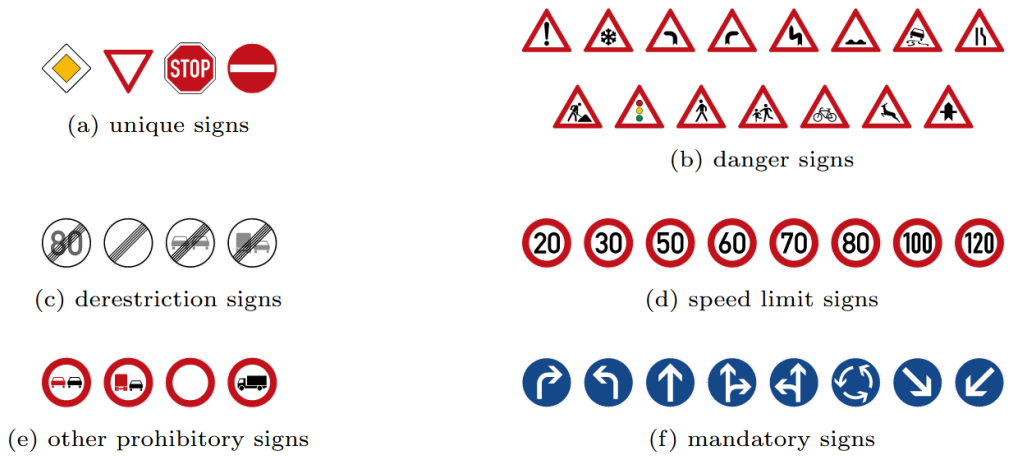
I will also verify the Groups constraint, as shown in the work of Flinkow *et al.* [30]. This property aims to guarantee groups of similar signs, such as speed limits, are predicted together, i.e. the total sum of a groups probability is either high or low. The way road signs have been grouped in shown in Figure 3.3

Unfortunately the original property relies on calculating the probability of each class using a Softmax layer. Softmax is unsupported by the vast majority of verifiers due to its complex non-linearity, and as such I have designed a new property with similar goals.

Instead of ensuring entire groups are predicted with either a high or low probability as in the original constraint, I will be verifying that top predictions of the model are members of the group an image belongs to. This is formally written as:

$$\forall i \in G_y, j \notin G_y : \text{Model}(x)_i > \text{Model}(x)_j$$

Where $G_y$ is the set of classes inside the group $x$ belongs to. This can be expanded to test the robustness of the model regarding this property by allowing $x$ to be a member of the epsilon ball around the original image.




(a) unique signs

(b) danger signs

(c) derestriction signs

(d) speed limit signs

(e) other prohibitory signs

(f) mandatory signs

FIGURE 3.3: Road sign Groups (reproduced from [30]).

# Chapter 4

# Implementation

## 4.1 Dataset

The dataset I am using is the GTSRB dataset [2], containing 43 classes of road signs, and a mixture of image sizes ranging from 20×20 to 200×200. In total there are 39000 labelled images.

Firstly I have converted the images from ".ppm" files to the PNG format, so they can be imported using PyTorch. To deal with the mixed image sizes and also reduce the strain of verification, I have scaled all images to 32×32 pixels as well to greyscale. These images were then normalised by the mean and standard deviation calculated from the dataset. The processed images are shown in Figure 4.1.

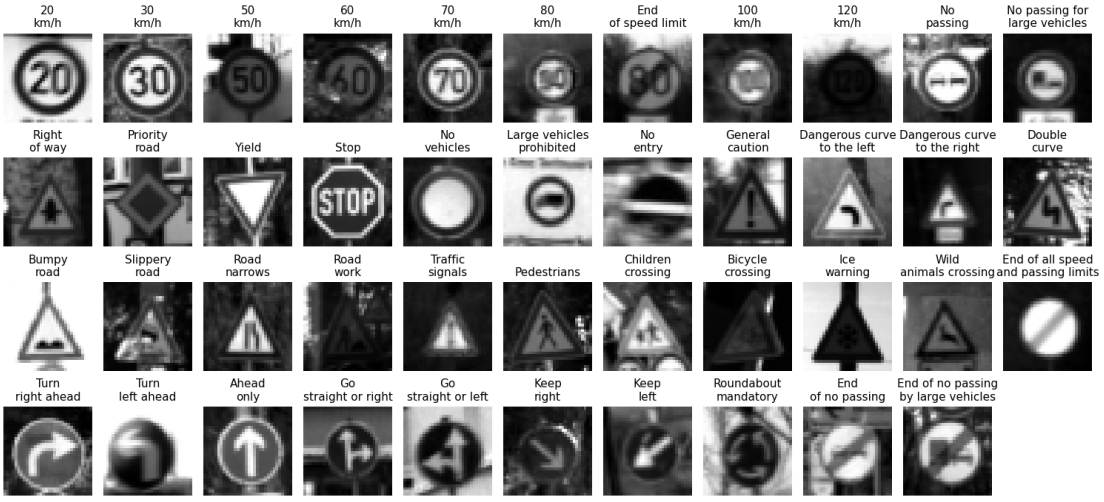



FIGURE 4.1: Processed images and their labels.

For my experiments I have used an 80 / 20 train-test split of the data, so I can gauge the performance of the model on unseen data and make sure it works.

## 4.2 Model

To implement the CNN, I used the PyTorch [31] library in Python. A summary of the model is shown in Figure 4.2.

| Layer | Output Shape | Parameters |
|---|---|---|
| Input | [256,1,32,32] | 0 |
| Conv2d-1 | [256, 6, 28, 28] | 156 |
| AvgPool2d-2 | [256, 6, 14, 14] | 0 |
| Conv2d-3 | [256, 16, 10, 10] | 2,416 |
| AvgPool2d-4 | [256, 16, 5, 5] | 0 |
| Linear-5 | [256, 120] | 48,120 |
| Linear-6 | [256, 80] | 9,680 |
| Linear-7 | [256, 43] | 3,483 |

FIGURE 4.2: The model's Architecture (For a batch of 256).

## 4.3 Training

Each model was trained for 100 epochs, with early stopping if no improvement occurred after 8 epochs. I tested three different optimisers in conjunction with adversarial training, results shown in Figure 4.3. From this I decided on the Adam optimiser, as while it performed very similarly to SGD, it is typically less sensitive to 'bad' hyperparameters, which is ideal when experimenting with different training methods.

| Optimiser | Precision | Recall | F1-Score |
|---|---|---|---|
| Adam | 0.933 | 0.927 | 0.928 |
| AdamW | 0.920 | 0.904 | 0.907 |
| SGD | 0.935 | 0.925 | 0.926 |

FIGURE 4.3: Metrics for different optimisers.

## 4.4 Robustness Training

Models were trained on batches comprised of 50% standard images, and 50% adversarial images, generated from the first half of the batch.

### 4.4.1 Generating Adversaries

To generate I used Projected Gradient Descent [17]. On occasion this method can falter, getting stuck at points of local maxima of the loss function, and can actually perform worse than FGSM [1] when using a smaller step size.

To combat this I have introduced momentum into the PGD equation as suggested by Wu *et al.* [32]. I've also added exponential decay of the step size, so as the algorithm converges changes will be finer.

After this, I used grid search to tune the parameters, attempting to maximise the average loss across the test dataset on a previously trained model.

Comparison results can be seen in 4.4, for adversaries generated with an epsilon of 10 (in the 0-255 colour space).

| **Method** | **Total Loss** |
|---|---|
| FGSM | 624 |
| PGD | 1080 |
| Improved PGD | 1190 |

FIGURE 4.4: Comparison of the total loss on adversarial images generated from the test set.

Examples of the adversarial images generated can be seen in 4.5. Concluding from these images, epsilon balls larger than 10 are unreasonable to verify, as the images start to become unrecognisable.

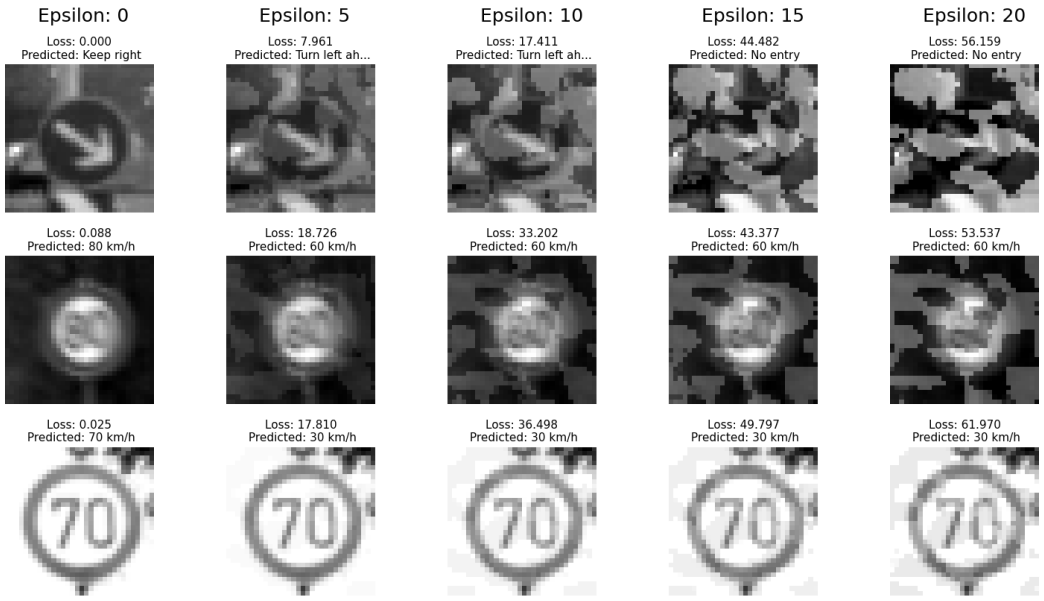



FIGURE 4.5: Adversarial images generated from the dataset.

## 4.5 Robustness Verification

Originally I planned to verify this property using the Vehicle language [13] and Marabou verifier [28], as Vehicle was designed to be much more user-friendly than other verifiers.

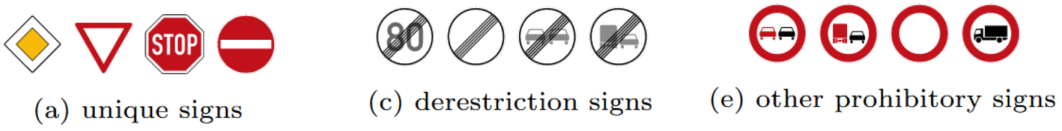
However, despite my attempts to make verification easier, such as using a small model and image size, my testing found it was very easy to run out of memory during verification. While this could have potentially been resolved by reducing the number of classes in the dataset, I instead opted to use AB-CROWN [22]–[27]. While it is less user-friendly compared to Vehicle, its utilisation of GPU acceleration makes it notably faster. It also importantly features a tuneable batch size parameter, which allows me to reduce the strain on memory.

For verification of this property, AB-CROWN has built in support and simply needs a config file specifying the details like the model, dataset and verification algorithm. Based on guidance from the AB-CROWN GitHub, I decided on using the Beta-CROWN solver.

As verifying the entire dataset would take an extremely long time, I have decided to verify only 5 images per class, for a total of 215 images.

## 4.6 Groups Constraint

Due to the increased complexity of the Groups constraint, and thus increased complexity of verification I have reduced the number of groups to just 3, with 12 classes between them. The remaining groups are shown in Figure 4.6.




(a) unique signs (c) derestriction signs (e) other prohibitory signs

FIGURE 4.6: The remaining groups for verification, reproduced from [30].

### 4.6.1 Training

Originally I planned to use the work of Flinkow *et al.* [30] to train my model. Using their work, the constraint can be defined as a fuzzy logic, and used as a loss function for training the model. Unfortunately even after reducing the groups, the cost of calculating the loss gradient of the fuzzy logic function is exceedingly expensive, taking about 40 minutes per epoch.

Instead, I have treated the group constraint as a multi label classification problem. In multi label classification the model is expected to output a probability for each label representing the likelihood a given input is a member of that class. For my purposes, this means the model is expected to output a high probability for all the members of a given road signs group, and low for the rest.

For this I will be using binary cross entropy loss. I can then also train for standard accuracy simultaneously, by combining the binary cross entropy loss and standard cross entropy loss with simple addition. As shown below, I also include additional weights to these loss functions allowing control over the balancing of the two goals.

$$\mathcal{L} = \alpha\mathcal{L}_{\text{CE}} + \beta\mathcal{L}_{\text{BCE}}$$

### 4.6.2 Verification

Once again I attempted to verify the property using Vehicle, as the majority of verifiers lack a convenient way of verifying more complex properties. As before the property was simply too difficult to verify using Marabou on my own hardware, and I ran out of memory.

As such I once again used AB-CROWN for verification. However, as AB-CROWN does not have a simple way of defining a unique constraint like this, I had to define it using the VNN-LIB format [33].

### 4.6.3 VNN-LIB Format

The VNN-LIB [33] format involves defining the problem as a series of constraints on the inputs and outputs of the network. The model must then be unable to satisfy these constraints in order to be proved correct for this property.

In my case this means defining the epsilon ball around the image as the upper and lower bound of each pixel (accounting for the min / max values a pixel can have, as well as the epsilon ball), as shown in Figure 4.7. For simplicity, the intensity values of each pixel are in the 0-255 colour space, but in reality these are normalised for the model.

This also means each image must have its own VNN-LIB file.

```
(assert (<= X_0 132))
(assert (>= X_0 122))

(assert (<= X_1 35))
(assert (>= X_1 25))

(assert (<= X_2 7))
(assert (>= X_2 0))
...
```

FIGURE 4.7: Example of input constraints for a ball of size 5.

The group constraint can then be formed as a constraint on the outputs of the model. As the goal is to disprove the constraints, this can be framed as a disjunction. The parser used by AB-CROWN is fairly rudimentary, designed specifically

```
(assert (or
    (and (>= Y_4 Y_0))
    (and (>= Y_5 Y_0))
    (and (>= Y_6 Y_0))
    ...
    (and (>= Y_4 Y_1))
    (and (>= Y_5 Y_1))
    (and (>= Y_6 Y_1))
    ...
```

FIGURE 4.8: Example of the output constraints, where the correct group contains labels: $\{0, 1, 2, 3\}$

for VNN Comp [21], and is hard coded to take two layers of an "and" or "or" operator. Hence, each comparison is wrapped in the "and" operation. An example is shown in Figure 4.8.

# Chapter 5

# Results

## 5.1 Robustness Property

My primary experiment in verifying the robustness constraint was the effect of the epsilon value used during adversarial training.

The verification scores below in Figure 5.1 were from verifying 215 samples (5 samples for each road sign) for a ball of radius 5. Verification score represents the percentage of images the model was able to satisfy the robustness property for.

I have also plotted the effects of adversarial training on the accuracy of the model. We can see a clear example of the "Robustness-accuracy trade-off", as suggested by Tsipras *et al.* [34]. As the training epsilon is increased, the model is provably more robust as shown by verification, and yet it still loses standard accuracy.

It's particularly interesting that despite the significant drop in accuracy, the verification score continues to increase, suggesting a significant enough boost in robustness to counteract the loss in overall accuracy. Ultimately in this case, such a significant trade off as in the case of 15 & 20 epsilon is antithetical to the original purpose of creating a model that is safer and more reliable. I might hypothesise that a sufficiently complex enough model to accurately classify adversaries within these larger epsilon balls could boast a significantly higher verification score, but verifying such a complex model would prove difficult.

It is worth noting that a substantially larger number of images timed out during verification for the 5 and 10 epsilon models. Timing out typically occurs when a sample is very close to the boundary of satisfying the constraint; being so close to

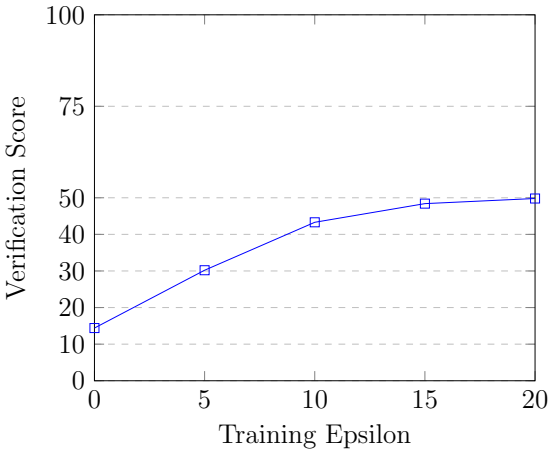



FIGURE 5.1: Comparison of how the training epsilon affects verification.

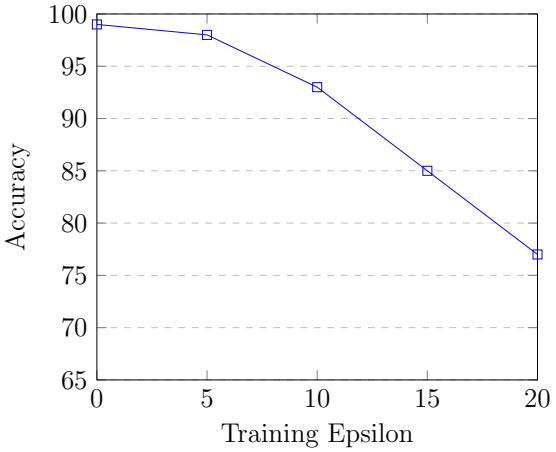



FIGURE 5.2: A display of the effects of training epsilon on model accuracy.

the boundary exponentially increases the cost of verification. As such the increase in timeouts is expected for models where the trained epsilon is so close to that of what is being verified, but it has the unfortunate result of significantly reducing the total verification score of these models (timeouts are treated as not satisfying the property).

For my experiments a 5-minute timeout was used. Unfortunately very few samples that took longer than 2 minutes were verified before timing out, so it is unlikely a longer timeout could improve results here.

| **Trained Epsilon** | **Total Timeouts** |
|---|---|
| 0 | 26 |
| 5 | 76 |
| 10 | 50 |
| 15 | 24 |
| 20 | 22 |

FIGURE 5.3: Number of timed out samples

Figure 5.1 shows a comparison of constraint security with the verification score. The large difference between these two metrics, suggests the adversaries being generated are far from optimal, potentially due to gradient masking [35], where as opposed to actually becoming more robust, adversarial training causes the model to reduce the "mask" its gradient, making it useless for generating adversaries.

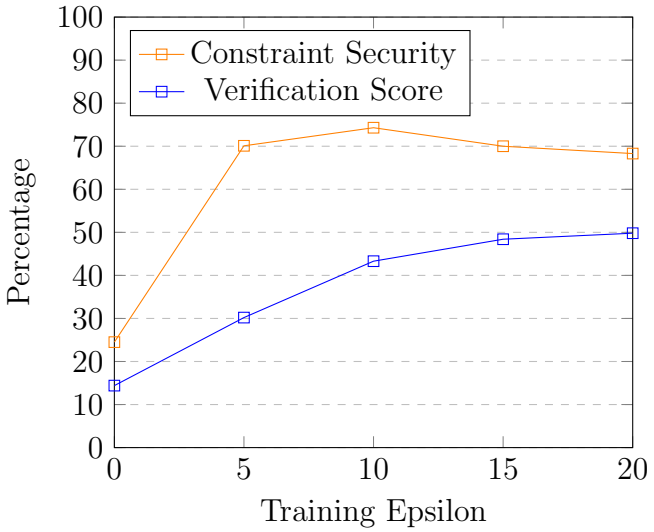



FIGURE 5.4: A plotting of constraint security compared to verification score.

One method to solve this problem might be to use a black box attack [16], where adversaries are generated on a non adversarially trained network, then used for adversarial training.

Lastly, choosing the model trained with 10 epsilon as a sensible trade-off of accuracy for robustness, I tested how well the model performs across a variety of epsilon balls, shown in Figure 5.1.

Despite being trained for 10 epsilon, the model is barely able to reach a 1% verification score. The model is clearly not complex enough to properly model robust classification for higher epsilon values.

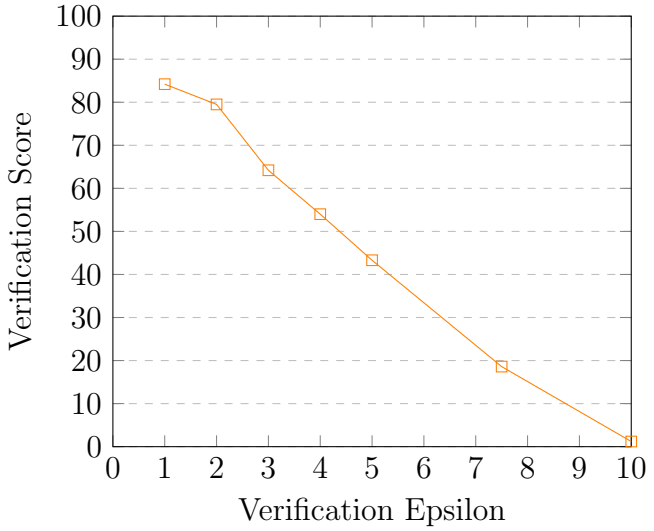



FIGURE 5.5: A plotting of constraint security compared to verification score.

## 5.2 Groups Constraint

For this constraint I have trained 3 models, a baseline, a model trained exclusively for the constraint (with adversarial training), and a model trained for both (with adversarial training). Based on my previous experiments on standard robustness, I have chosen an epsilon of 10 as a reasonable value for training balancing accuracy and robustness. My experiments verified 10 samples from each class, for a total of 120 samples.

As expected, them model trained explicitly for the constraint performs best, at great cost to standard accuracy. More notable is the lack of deficiency in the hybrid model for standard accuracy. This suggests that with only 12 labels such a small model is able to quite accurately capture the goals of standard accuracy, robustness and the groups constraint all in one. Following this if I had more time, a model trained with a greater weighting towards the groups constraint would very likely perform better in verification.

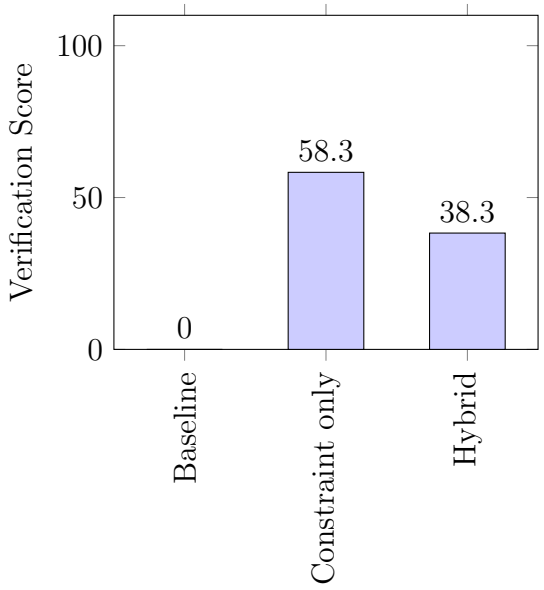



FIGURE 5.6: A comparison of model performances in verification.

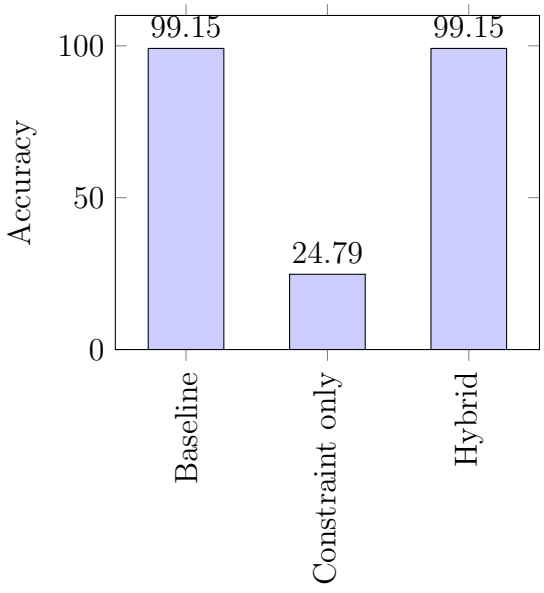



FIGURE 5.7: A comparison of model performance for standard accuracy.

Following my previous experiments I have also verified both the constraint only and hybrid model at on epsilon balls. Unsurprisingly the constraint only model continues to perform best. More notable is that even at an epsilon of 10, both models manage to stay above a 10% verification score, in contrast to my scores in

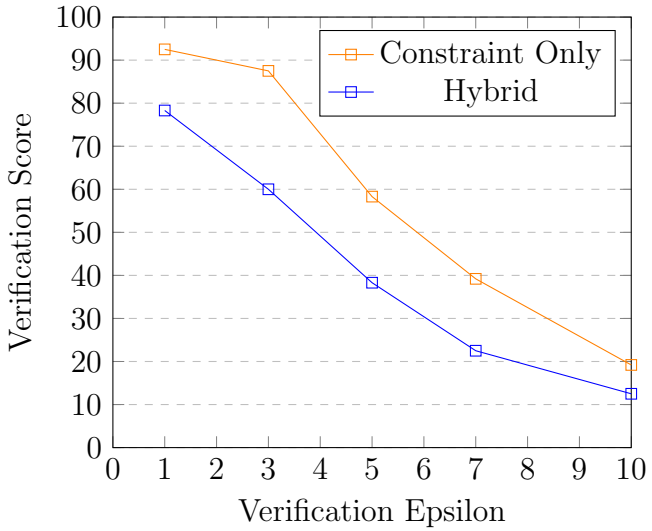



FIGURE 5.8: A plotting of model performance across various verified epsilon balls.

Figure 5.5, where they dropped to just 1%. This is likely explained by the reduced number of classes in the problem.

# Chapter 6

# Testing

## 6.1 Model Testing

To ensure my models worked, I recorded a variety of metrics, such as the validation loss at each epoch during training, and metrics like accuracy, precision and recall after training.

Tracking the loss on a separate validation set allowed me to spot overfitting, and ensure the training loop was working as expected. Similarly, accuracy metrics allowed me to gauge the overall performance of the model.

I also created visuals like in Figure 4.5, as a more human-readable way of testing performance.

## 6.2 Verification

To ensure I have correctly set up the verification experiments, I cross-checked the results with adversarial attacks inside the same epsilon ball.

While this cannot completely guarantee I have a correct setup, if an adversarial was able to prove a property didn't hold on an image, then verification must similarly show this.

To test for proper encoding of images into the VNN-LIB format, I wrote a simple parser to convert the constraints back into an image to be compared with the original.

# Chapter 7

# Project Management

Overall I found I did at times stray from my project plan. For semester 1 I found learning TensorFlow while implementing the model caused me to take a week or so longer than planned.

In semester 2 I discovered troubles with verification using Marabou [28], and decided to switch to AB-CROWN [22]–[27]. This switch required me to also move my work to PyTorch [31], which was an unexpected cost. Moving forward I had further concerns about verifying the groups constraint, leading me to implement and test this before I programmed constraint training.

## 7.1 Gantt Chart

After semester 1, I had a better knowledge of what my project was to include, and created an updated Gantt chart to give me a more accurate high level overview of the work to be done.

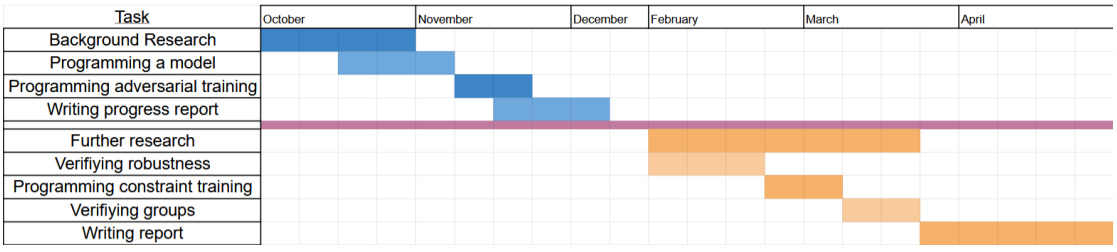



FIGURE 7.1: Updated Gantt Chart.

## 7.2 Risk Assessment

In order to better manage my project I have created a risk assessment, in order to help me anticipate problems and design solutions.

| **Risk** | **Probability** | **Severity** | **Score** | **Mitigation** |
|---|---|---|---|---|
| Work taking longer than time allocated in plan | 3 | 3 | 9 | I have allocated hopefully more than enough time for each section so if one overruns I can still reasonably complete the next section in time. |
| Losing work / Project becoming corrupted | 1 | 5 | 5 | I have backed up my report and code to GitHub. |
| Illness | 3 | 3 | 6 | Again enough time should be allocated to each section that I can complete it in good time. |
| Other modules taking up my time | 3 | 4 | 12 | My time allocations should be generous enough to account for this. |
| Verifier crashes during verification | 3 | 4 | 12 | AB-CROWN fortunately saves every time a sample is verified, so I can stitch results back together. |

FIGURE 7.2: A Risk assessment for my project.

# Chapter 8

# Conclusion

My overall goals with this project was to create classifiers with proven guarantees of robustness on the GTSRB dataset. While the overall verification score of my models was not close to 100%, my results show that significant increases in robustness can be achieved using the methods shown in this paper.

## 8.1 Future Work

There is much that could be done in extension to the work here. With greater computing power larger models like ResNet [36] could be trained adversarially, and would likely be sufficiently complex to capture robust classification of road signs.

As noted my work also had problems with suboptimal adversarial attacks. Stronger adversaries could be generated with techniques like Auto PGD [37], or non-gradient based techniques like BaB attacks [38].

Furthermore, techniques like GradNorm [39] could more effectively train models for standard accuracy as well as a constraint.

Finally, there are many more properties that could be trained and verified for, like Lipschitz robustness [12].

# Bibliography


[1] I. J. Goodfellow, J. Shlens, and C. Szegedy. "Explaining and Harnessing Adversarial Examples." arXiv: 1412.6572, Accessed: Nov. 23, 2024. [Online]. Available: http://arxiv.org/abs/1412.6572, pre-published.

[2] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural Networks*, Selected Papers from IJCNN 2011, vol. 32, pp. 323–332, Aug. 1, 2012, ISSN: 0893-6080. DOI: 10.1016/j.neunet.2012.02.016. Accessed: Dec. 4, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608012000457.

[3] T. Flinkow, B. A. Pearlmutter, and R. Monahan. "Comparing Differentiable Logics for Learning with Logical Constraints." arXiv: 2407.03847 [cs], Accessed: Dec. 9, 2024. [Online]. Available: http://arxiv.org/abs/2407.03847, pre-published.

[4] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, ISSN: 1558-2256. DOI: 10.1109/5.726791. Accessed: Dec. 3, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/726791.

[5] M. S. Nixon and A. S. Aguado, "Chapter 4 - Low-level feature extraction (including edge detection)," in *Feature Extraction & Image Processing for Computer Vision (Third Edition)*, M. S. Nixon and A. S. Aguado, Eds., Oxford: Academic Press, Jan. 1, 2012, pp. 137–216, ISBN: 978-0-12-396549-3. DOI: 10.1016/B978-0-12-396549-3.00004-5. Accessed: Dec. 3, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123965493000045.

[6] K. O'Shea and R. Nash. "An Introduction to Convolutional Neural Networks." arXiv: 1511.08458 [cs], Accessed: Dec. 4, 2024. [Online]. Available: http://arxiv.org/abs/1511.08458, pre-published.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 24, 2017, ISSN: 0001-0782. DOI: 10.1145/3065386. Accessed: Dec. 1, 2024. [Online]. Available: https://dl.acm.org/doi/10.1145/3065386.

[8] C. Szegedy *et al.* "Going Deeper with Convolutions." arXiv: 1409.4842 [cs], Accessed: Dec. 3, 2024. [Online]. Available: http://arxiv.org/abs/1409.4842, pre-published.

[9] M. Abadi *et al.* "TensorFlow: A system for large-scale machine learning." arXiv: 1605.08695 [cs], Accessed: Dec. 4, 2024. [Online]. Available: http://arxiv.org/abs/1605.08695, pre-published.

[10] C. Szegedy *et al.* "Intriguing properties of neural networks." arXiv: 1312.6199 [cs], Accessed: Nov. 26, 2024. [Online]. Available: http://arxiv.org/abs/1312.6199, pre-published.

[11] H. B. Braiek and F. Khomh. "Machine Learning Robustness: A Primer." arXiv: 2404.00897 [cs], Accessed: Nov. 27, 2024. [Online]. Available: http://arxiv.org/abs/2404.00897, pre-published.

[12] M. Casadio *et al.* "Neural Network Robustness as a Verification Property: A Principled Case Study." arXiv: 2104.01396 [cs], Accessed: Nov. 25, 2024. [Online]. Available: http://arxiv.org/abs/2104.01396, pre-published.

[13] M. L. Daggitt, W. Kokke, R. Atkey, N. Slusarz, L. Arnaboldi, and E. Komendantskaya. "Vehicle: Bridging the Embedding Gap in the Verification of Neuro-Symbolic Programs." arXiv: 2401.06379 [cs], Accessed: Nov. 27, 2024. [Online]. Available: http://arxiv.org/abs/2401.06379, pre-published.

[14] M. Fischer, M. Balunovic, D. Drachsler-Cohen, T. Gehr, C. Zhang, and M. Vechev, "DL2: Training and Querying Neural Networks with Logic," *Proceedings of Machine Learning Research*, 2019.

[15] C. Shorten and T. M. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning," *Journal of Big Data*, vol. 6, no. 1, p. 60, Jul. 6, 2019, ISSN: 2196-1115. DOI: 10.1186/s40537-019-0197-0. Accessed: Nov. 27, 2024. [Online]. Available: https://doi.org/10.1186/s40537-019-0197-0.

[16] F. Tramèr, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel. "The Space of Transferable Adversarial Examples." arXiv: 1704.03453 [stat], Accessed: Nov. 24, 2024. [Online]. Available: http://arxiv.org/abs/1704.03453, pre-published.

[17] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. "Towards Deep Learning Models Resistant to Adversarial Attacks." arXiv: 1706.06083 [stat], Accessed: Nov. 23, 2024. [Online]. Available: http://arxiv.org/abs/1706.06083, pre-published.

[18] G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer. "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks." arXiv: 1702.01135 [cs], Accessed: Dec. 10, 2024. [Online]. Available: http://arxiv.org/abs/1702.01135, pre-published.

[19] L. Pulina and A. Tacchella, "An Abstraction-Refinement Approach to Verification of Artificial Neural Networks," in *Computer Aided Verification*, T. Touili, B. Cook, and P. Jackson, Eds., red. by D. Hutchison *et al.*, vol. 6174, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 243–257, ISBN: 978-3-642-14294-9. DOI: 10.1007/978-3-642-14295-6_24. Accessed: Dec. 10, 2024. [Online]. Available: http://link.springer.com/10.1007/978-3-642-14295-6_24.

[20] L. Datta. "A Survey on Activation Functions and their relation with Xavier and He Normal Initialization." arXiv: 2004.06632 [cs], Accessed: Dec. 8, 2024. [Online]. Available: http://arxiv.org/abs/2004.06632, pre-published.

[21] C. Brix, S. Bak, C. Liu, and T. T. Johnson. "The Fourth International Verification of Neural Networks Competition (VNN-COMP 2023): Summary and Results." arXiv: 2312.16760 [cs], Accessed: Dec. 10, 2024. [Online]. Available: http://arxiv.org/abs/2312.16760, pre-published.

[22] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel. "Efficient Neural Network Robustness Certification with General Activation Functions." arXiv: 1811.00866 [cs], Accessed: Dec. 10, 2024. [Online]. Available: http://arxiv.org/abs/1811.00866, pre-published.

[23] K. Xu *et al.* "Automatic Perturbation Analysis for Scalable Certified Robustness and Beyond." arXiv: 2002.12920 [cs], Accessed: Dec. 10, 2024. [Online]. Available: http://arxiv.org/abs/2002.12920, pre-published.

[24] K. Xu *et al.* "Fast and Complete: Enabling Complete Neural Network Verification with Rapid and Massively Parallel Incomplete Verifiers." arXiv: 2011.13824 [cs], Accessed: Dec. 10, 2024. [Online]. Available: http://arxiv.org/abs/2011.13824, pre-published.

[25] S. Wang *et al.* "Beta-CROWN: Efficient Bound Propagation with Per-neuron Split Constraints for Complete and Incomplete Neural Network Robustness Verification." arXiv: 2103.06624 [cs], Accessed: Dec. 10, 2024. [Online]. Available: http://arxiv.org/abs/2103.06624, pre-published.

[26] Z. Shi, Q. Jin, Z. Kolter, S. Jana, C.-J. Hsieh, and H. Zhang. "Neural Network Verification with Branch-and-Bound for General Nonlinearities." arXiv: 2405.21063 [cs], Accessed: Dec. 10, 2024. [Online]. Available: http://arxiv.org/abs/2405.21063, pre-published.

[27] H. Zhang *et al.* "General Cutting Planes for Bound-Propagation-Based Neural Network Verification." arXiv: 2208.05740 [cs], Accessed: Dec. 10, 2024. [Online]. Available: http://arxiv.org/abs/2208.05740, pre-published.

[28] H. Wu *et al.* "Marabou 2.0: A Versatile Formal Analyzer of Neural Networks." arXiv: 2401.14461 [cs], Accessed: Nov. 27, 2024. [Online]. Available: http://arxiv.org/abs/2401.14461, pre-published.

[29] A. Lemesle, J. Lehmann, and T. L. Gall. "Neural Network Verification with PyRAT." arXiv: 2410.23903 [cs], Accessed: Dec. 10, 2024. [Online]. Available: http://arxiv.org/abs/2410.23903, pre-published.

[30] T. Flinkow, B. A. Pearlmutter, and R. Monahan, "Comparing Differentiable Logics for Learning with Logical Constraints," *Science of Computer Programming*, p. 103 280, Mar. 7, 2025, ISSN: 0167-6423. DOI: 10.1016/j.scico.2025.103280. Accessed: Mar. 11, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016764232500019X.

[31] A. Paszke *et al.* "PyTorch: An Imperative Style, High-Performance Deep Learning Library." arXiv: 1912.01703 [cs], Accessed: Apr. 20, 2025. [Online]. Available: http://arxiv.org/abs/1912.01703, pre-published.

[32] F. Wu, R. Gazo, E. Haviarova, and B. Benes. "Efficient Project Gradient Descent for Ensemble Adversarial Attack." arXiv: 1906.03333 [cs], Accessed: Nov. 24, 2024. [Online]. Available: http://arxiv.org/abs/1906.03333, pre-published.

[33] D. Guidotti, S. Demarchi, A. Tacchella, and L. Pulina, "The VNN-LIB standard for benchmarks 2022," Nov. 11, 2022. [Online]. Available: https://www.vnnlib.org/.

[34] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry. "Robustness May Be at Odds with Accuracy." arXiv: 1805.12152 [stat], Accessed: Nov. 24, 2024. [Online]. Available: http://arxiv.org/abs/1805.12152, pre-published.

[35] A. Kurakin *et al.* "Adversarial Attacks and Defences Competition." arXiv: 1804.00097 [cs], Accessed: Feb. 27, 2025. [Online]. Available: http://arxiv.org/abs/1804.00097, pre-published.

[36] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition." arXiv: 1512.03385 [cs], Accessed: Apr. 29, 2025. [Online]. Available: http://arxiv.org/abs/1512.03385, pre-published.

[37] F. Croce and M. Hein. "Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks." arXiv: 2003.01690 [cs], Accessed: Apr. 29, 2025. [Online]. Available: http://arxiv.org/abs/2003.01690, pre-published.

[38] H. Zhang *et al.*, "A Branch and Bound Framework for Stronger Adversarial Attacks of ReLU Networks,"

[39] Z. Chen, V. Badrinarayanan, C.-Y. Lee, and A. Rabinovich. "GradNorm: Gradient Normalization for Adaptive Loss Balancing in Deep Multitask Networks." arXiv: 1711.02257 [cs], Accessed: Apr. 29, 2025. [Online]. Available: http://arxiv.org/abs/1711.02257, pre-published.

# Chapter 9

# Appendix

## 9.1 Design Archive

- Robustness Folder
  - pytorch greyscale.ipynb : Used for experimenting
  - base_model.py : Used to train models
  - model_perf.ipynb : Used to get model statistics
  - train.sh : Used to train several models
  - verify.sh : Used to verify all models in models directory
  - oneshot.sh : Used to verify 10 epsilon model on several epsilon balls
  - models folder : Stores PyTorch models
- Superclass Folder
  - superclass.ipynb : Used to train the models
  - model_performance.ipynb : Used to obtain model performances
  - verify.sh : Verifies models
  - vnnlib.ipynb : Used to create VNNLIB dataset
  - dataset.py : Used to import dataset
  - models folder : Stores models
- alpha-beta-CROWN
  - *.pkl files : Results from alpha-beta-CROWN, stored as python dictionaries

  - custom folder : Stores model and dataset definitions for verification
  - exp_configs : Contains config files for verification
  - Note : AB-CROWN has been removed for hand in (limited file size). Must be reinstalled to replicate experiments.

- dataset folder : Used to store dataset (removed due to hand in size limits, GTSRB dataset is expected to be at dataset/GTSRB/Training)
- read_log.ipynb : Used to read AB-CROWN results