# Contents

# List of Figures

# List of Tables

# 1

# Introduction

## 1.1 Cyber-Physical Systems

Introduce the concept of safety critical, CPS. What makes a system safety critical?

## 1.2 Artificial Neural Networks

Introduce AI, more specifically ANNs. What they can do?

## 1.3 Synchronous Programming

What are synchronous languages, why are they used, where are they used?

## 1.4 Contribution

Creating methods of implementing ANNs in safety critical systems using synchronous semantics.

## 1.5 Thesis Structure

...?

# 2
# Background

## 2.1 Artificial Neural Networks

### 2.1.1 Machine Learning

What is machine learning. Where is it used. Some examples.

### 2.1.2 Efficacy of Machine Learning

How does machine learning (ANNs) improve systems these days?

### 2.1.3 Structure of an Artificial Neural Network

How does an ANN run, how does it train, what does it look like.

### 2.1.4 Safety of Artificial Neural Networks

Safety drawbacks of ANNs: Unpredictable Hard to verify Slow Etc

## 2.2 Synchronous Languages

### 2.2.1 Synchronous Semantics

Briefly explain synchronous semantics: what is synchronous programming and how does it work?

### 2.2.2 Safety of Synchronous Languages

What guarantees does synchronous programming provide? Determinism Causality Easy to formalise Etc

### 2.2.3 Timing Correctness

Time predictability.

### 2.2.4 Functional Correctness

Runtime Enforcement

## 2.3 Time Predictable Cyber-Physical Systems

### 2.3.1 Worst Case Execution Time

Explain what WCET is in safety critical systems. Why is it important for deadlines to be met and how does this relate to WCET?

### 2.3.2 Worst Case Reaction Time

WCRT vs WCET. Throughput vs Time.

### 2.3.3 Timing Techniques

What timing techniques exist for synchronous and asynchronous systems? Which of these are relevant to our work?

### 2.3.4 Patmos Time Predictable Processor

Brief description of Patmos and why we use it.

## 2.4 Verification of Artificial Neural Networks

This section looks into ANN verification methods and why they are good/bad.

### 2.4.1 The Problem of Artificial Neural Network Verification

ANN verification is an issue. Why?

### 2.4.2 Existing Techniques for Verifying ANNs

Rundown of existing techniques of ANN verification.

### 2.4.3 Limitations of Artificial Neural Network Verification

Describe where verification is limited and why. Where can it do better?

## 2.5 Run-time Enforcement

### 2.5.1 Run-time Assurance

What is RA and why is it useful? Formally defined.

### 2.5.2 Run-time Monitoring

How does this relate and improve on RA?

### 2.5.3 Run-time Enforcement

Benefits? Synchronous vs asynchronous systems.

### 2.5.4 Safety (Timed) Automata

These are used in our run-time enforcement techniques. Discuss these and how they work.

## 2.6 Summary

Summary of above.

# 3

# Synchronous Neural Networks

## 3.1 Synchronous Neural Networks

### 3.1.1 Fundamentals of Artificial Neural Networks

Explain what is needed in an ANN library.

### 3.1.2 Architecture of Synchronous Neural Networks

Describe how SNNs run and how they work as an ANN library.

### 3.1.3 Formalalisation of Synchronous Neural Networks

This subsection provides a formal definition for feed forward SNNs

### 3.1.4 Implementation of Synchronous Neural Networks

Describe how to use the SNN library. Describe both the C and Esterel aspects of SNNs.

## 3.2 Meta Neural Networks

This section describes how MNNs relate to SNNs

### 3.2.1 Architecture of Meta Neural Networks

Explain how MNNs are created and how they function

### 3.2.2 Formalisation of Meta Neural Networks

Provide the formal definition for MNNs

### 3.2.3 Implementation of Meta Neural Networks

Explain how MNNs are implemented and why they are useful. Maybe give some basic examples.

## 3.3 Timing Analysis of Synchronous Neural Networks

### 3.3.1 Patmos

Brief description of the Patmos used in timing analysis.

### 3.3.2 Time Predictable Software for Patmos

What is required of software to be time analysable on Patmos?

### 3.3.3 Timing of Meta Neural Networks

Describe how the timing of MNNs was accomplished.

## 3.4 A Case Study for Time Predictable Synchronous Neural Networks

### 3.4.1 An Energy Storage System for an Electric Vehicle Charging Station

Cite Kalpesh's paper and describe the system we are using as our case study. Why it is unsafe and how we can make it better.

### 3.4.2 Energy Storage System and Machine Learning

Describe the role of machine learning for the ESS and how the system can be improved. Describe the training methods used to train the ANNs, and elaborate on how the ANNs work for this system.

### 3.4.3 Synchronous Neural Network in an ESS

Methodology behind implementing the C and Esterel code for the trained ANN.

### 3.4.4 Timing Analysis of the ESS

Explain how timing was important for this system and how it can improve the safety of this system. WCET and WCRT.

## 3.5 ESS Case Study Results

Results of the ESS case study.

## 3.6 Synchronous Neural Network Timing Benchmark Results

Results of the benchmarks

### 3.6.1 RABBIT Game

### 3.6.2 Adder

### 3.6.3 AI-BRO

### 3.6.4 XOR

## 3.7 Summary

Summary of everything previously mentioned in this chapter

## 3.8 Discussion

Discussion of the results of the case study and benchmarks. Discuss methodology and results.

# 4

# Runtime Enforcement of Synchronous Neural Networks

## 4.1 Runtime Enforcement of Safety Critical Systems

Introduce the concept of runtime enforcement for safety critical systems.

### 4.1.1 Safety Automata

Explain how timed (safety) automata are used as policies for runtime enforcement. Why these are good.

### 4.1.2 Synchronous Runtime Enforcement

How does RTE works with synchronous systems and what are the benefits.

## 4.2 Runtime Enforcement of Synchronous Neural Networks

Section showing how we are using RTE with SNNs.

### 4.2.1 Safe Synchronous Neural Networks

Introduce the concept of combining SNNs and RTE to create a safe SNN

### 4.2.2 Semantics of Safe Synchronous Neural Networks

SSNN formal semantics in detail?

### 4.2.3 Safe Meta Neural Networks

How does the use of RTE expand to MNNs?

### 4.2.4 Semantics of Safe Meta Neural Networks

SMNN formal semantics in detail?

### 4.2.5 Timing of Safe Synchronous Neural Networks

Briefly discuss how these can be timed, etc.

## 4.3 A Case Study for Safe Synchronous Neural Networks

This is the created AV system with runtime enforcement.

### 4.3.1 Autonomous Vehicles

What are AVs, where are they used, why are they dangerous?

### 4.3.2 Safety Autonomous Vehicle Systems

How is this system unsafe? What did it do to make it unsafe.

### 4.3.3 Runtime Enforcement of Autonomous Vehicles Systems

Where can RTE be introduced to AV systems. What are its limitations? Where does our work take RTE of this system.

### 4.3.4 Architecture of an Autonomous Vehicle Simulation

We created a simulation of an AV system. What was done to ensure they were as similar as possible.

### 4.3.5 Safety Policy for an Autonomous Vehicle Simulation

The enforced safety policy for the above system.

## 4.4 Autonomous Vehicle Case Study Results

Results of the AV case study.

## 4.5 Safe Synchronous Neural Network Benchmark Results

Results of the above-mentioned benchmarks

### 4.5.1 RABBIT Game

### 4.5.2 Energy Storage System

## 4.6 Summary

Summary of everything previously mentioned in this chapter

## 4.7 Discussion

Discussion of the results of the case study and benchmarks. Discuss methodology and results.

# 5

# Runtime Verification of Synchronous Neural Networks

## 5.1 Verification of Deep Artificial Neural Networks

Deep Artificial Neural Networks (ANNs) [8], ANNs with large, complex inputs and a large, complex layer structure and multiple layers, are hard to verify due to their complex nature [5]. Lots of work has been put into the verification of ANNs, but the results yielded from this work have many limitations and are generally time consuming.

Previously, this thesis introduced the concept of using runtime enforcement to dynamically "verify" an ANN as it runs. While this works very well for simple, feed forward ANNs, this does not extend to more complicated Convolutional Neural Networks (CNNs).

### 5.1.1 Runtime Enforcement of Deep Artificial Neural Networks

Runtime Enforcement, as a solution to the verification of ANNs and introduced in the previous chapter, only works when the inputs to the ANN are known. This works very well with feed forward ANNs, as their inputs are, generally, defined and known. Take the AI-BRO ANN discussed in Section **??**; this ANN has a vector of defined inputs representing the objects "seen" by the vehicle. Since dangerous situations are defined by both inputs and outputs, the outputs of this ANN can be enforced because both the inputs and outputs are known at all times.

Complex ANN inputs (such as images for CNNs) are unknown, and the purpose of the ANNs is to classify said inputs. Since the inputs cannot be defined as something recognisable, the decision made by the ANN cannot be classified as dangerous because a dangerous situation is defined by known inputs and outputs. Take a CNN that classifies pedestrians; we are using the CNN because the software cannot recognise whether the input image shows a pedestrian or not. If the pedestrian is misclassified as a tree, the system has no way to know this based off of the inputs to the CNN alone. Thus, behaviour of ANNs with complex inputs (most commonly classification ANNs) cannot be enforced using only the inputs to the ANN.

The solution proposed is to use runtime enforcement as a means to safely combine the outputs from various sensors in the system and use these to enforce the outputs of the ANN.

### 5.1.2 Input Perturbation for Deep Artificial Neural Networks

This chapter also tackles another problematic aspect of using CNNs: input perturbations. A group showed that very slight modifications to the input image of a Convolutional Neural Network (CNN), such as the discolouration of a few pixels, could cause the image to be misclassified [5]. A CNN can train to high accuracy, e.g. 99.99%, on the training set, but simple perturbations to the CNN's input can lead to drastically reduced accuracy

once the CNN has been deployed. This thesis shows two methods of increasing the prediction accuracy of an CNN affected by input perturbations: the first is the above-mentioned sensor fusion with runtime enforcement and the other is using a Meta Neural Network (MNN) ensemble of different CNNs, each classifying a different aspect of the input image and congregating to make a decision as a whole.

### 5.1.3 Sensor Fusion and Runtime Verification for Deep Artificial Neural Networks

Sensor fusion is a highly researched topic for increasing the accuracy of object detection CNNs [11]. Sensor fusion is the combination of two or more different sensor types to increase the overall sensor detection accuracy of the system. Where Autonomous Vehicles (AVs) are concerned safety is of utmost importance, making this the biggest area of research for sensor fusion using CNNs.

The proposed approach to safe deep ANNs combines runtime enforcement and sensor fusion to safely increase the detection accuracy of CNNs in AVs. The sensors in question are a 360◦ Light Detection and Ranging (LiDAR) sensor and three front-facing cameras. The sensor outputs are combined synchronously using a synchronous runtime enforcer, however the runtime enforcer does not enforce the outputs of the detection system. Rather, the runtime enforcer verifies the integrity of the detected outputs, and in the case of a possible misclassification the system is put into a safe mode where control of the vehicle is returned to the driver. This type of runtime enforcement has been termed as “runtime verification”. Using a Safe Synchronous Neural Network (SSNN) as the CNN in the AV system, the system is kept synchronous, time predictable and easy to formalise.

Additionally, to reduce the impact of perturbations to the CNN’s inputs, the SSNN used in the system is actually a MNN composed of three MNN ensembles, each with three synchronous CNNs. This MNN aims to greatly increase the prediction accuracy by splitting the prediction process amongst multiple, different CNNs such that the weakness of each CNN is addressed by the other CNNs.

## 5.2 A Object Detection Case Study for Perturbed Inputs

Research in Autonomous Vehicles (AVs) systems is rapidly expanding, with companies such as Uber and Tesla the biggest contributors to this field. With the growing use of AVs comes an increase in accidents related to these vehicles. The majority of these accidents have one thing in common: a misclassification occurred right before the accident. Input perturbations greatly decrease accuracy, thereby increasing the chance of a misclassification and, by proxy, the chance of an accident.

### 5.2.1 Object Misclassification in Autonomous Vehicle Systems

The inspiration for this case study was taken from the more recent Uber and Tesla AV accident, such as [3]. In these accidents, a misclassification, or series thereof, precedented the crash. In the case of the Uber accident [3] a pedestrian crossing the road (at night) was misclassified a minimum of three times before the accident occurred.

Theoretically, this accident could have been prevented in two ways: the first being that no misclassifications occurred and the second being that the system recognised the misclassifications sooner and acted accordingly. The misclassifications could have occurred because it was night-time and the input(s) of the pedestrian were perturbed, or the misclassification could have occurred in that 0.001% of misclassifications. Either way, a system can be implemented to increase prediction accuracy regardless of the state of the inputs, perturbed or not. Secondly, by implementing a runtime enforcer and a safe, timed automaton as the safety policy misclassifications can be detected quickly and effectively.

### 5.2.2 Simulation of an Autonomous Vehicle Object Detection System

Replicating an AV system, such as Uber's or Tesla's, would be impossible; it is expensive to purchase the technology such as LiDAR and the machine vision cameras, and the time taken to implement the system as a single researcher is also infeasible. As such, a simulation of the AV was made to try and capture the intricacies of such a system where safety is concerned, while showing the efficacy of the proposed techniques.

## 5.3 Simulating a Traffic Sign Recognition AV System

The system designed for this case study was made to reflect a Autonomous Vehicle (AV) and its object detection mechanisms. The system used multiple techniques to tackle the inherent issues of the AV system, i.e. weakness to perturbed inputs and misclassification of detected objects. The system's sensors included an overhead, 360° Light Detection and Ranging (LiDAR) apparatus, and a single set of front-facing cameras. Using only front-facing cameras was sufficient to prove the efficacy of this solution, however it is to be noted that AV systems generally use cameras facing multiple, different directions, so that the controller can make properly informed decisions. The architecture of the system used can be seen in Figure 5.3.

### 5.3.1 Architecture of a Traffic Sign Recognition AV System

Utilising synchronous semantics [1], a Meta Neural Network (MNN), containing three other MNN ensembles, was created. A MNN is structure containing at least one Safe Synchronous Neural Network (SSNN) [7] synchronously combined with any number of other functional blocks [7]. Each of the three MNN ensembles were used to classify shape, colour and object type of each object being output by the camera. Each ensemble synchronously combined the outputs of three different convolutional SSNNs, providing increased prediction accuracy for shape, colour and object type. Each SSNN was implemented in the synchronous language Esterel [2]. Synchronous programming guarantees code that is deterministic, causal and bounded. Additionally, the calculation of Worst Case Execution Time (WCET) and Worst Case Reaction Time (WCRT) is enabled by using Esterel with C. These ensembles ran in synchronous concurrency with each other, forming a single, larger MNN. The outputs of each ensemble were then combined into a batch of outputs forming the *predicted output*.

### 5.3.2 Sensor Fusion for a Traffic Sign Recognition AV System

Two sensors were used for the purpose of sensor fusion; LiDAR and cameras. The LiDAR controller for this system was accurate 93% of the time [10], to closely simulate a system using real LiDAR. The simulated camera outputs consisted of test images from both the Visual Object Classes (VOC) 2012 [4] and German Traffic Sign Recognition Benchmark (GTSRB) [9] datasets, in a combination of people, vehicles and various traffic signs. The LiDAR and camera outputs were handled by different parts of the controller. The camera outputs were fed into a MNN (see Figure 5.1) where they were classified by shape, colour and object type. Sensor fusion happened after classification occurred and was done using an enforcer during runtime.

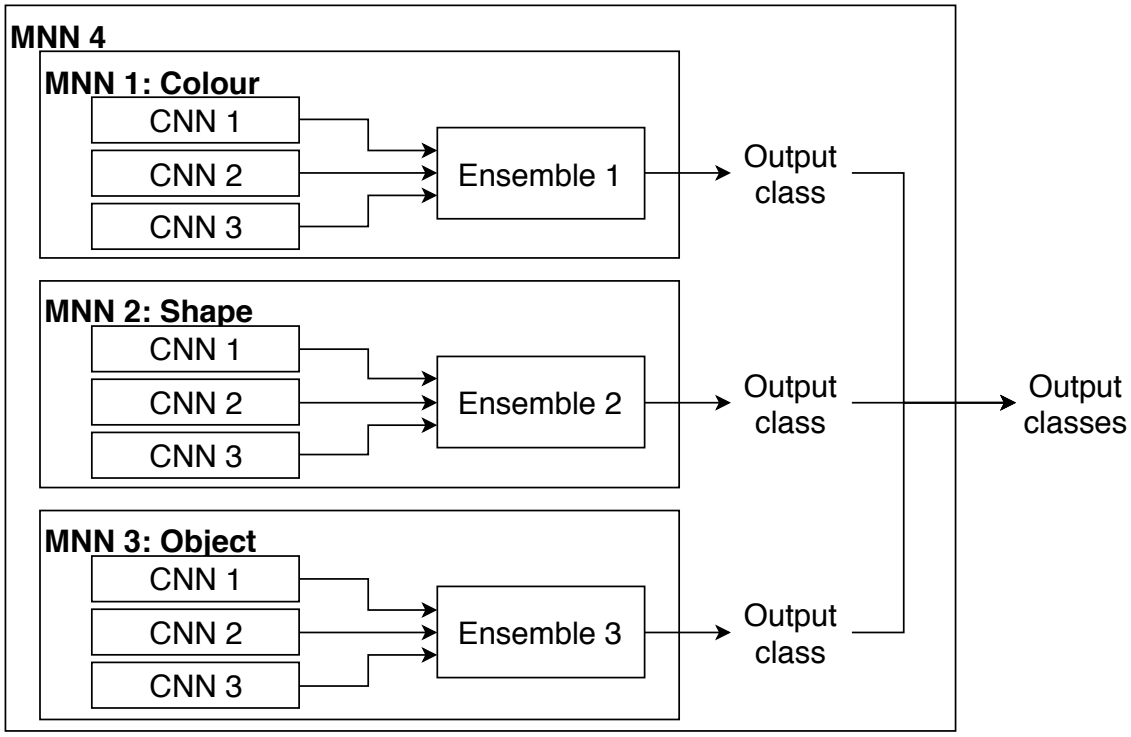


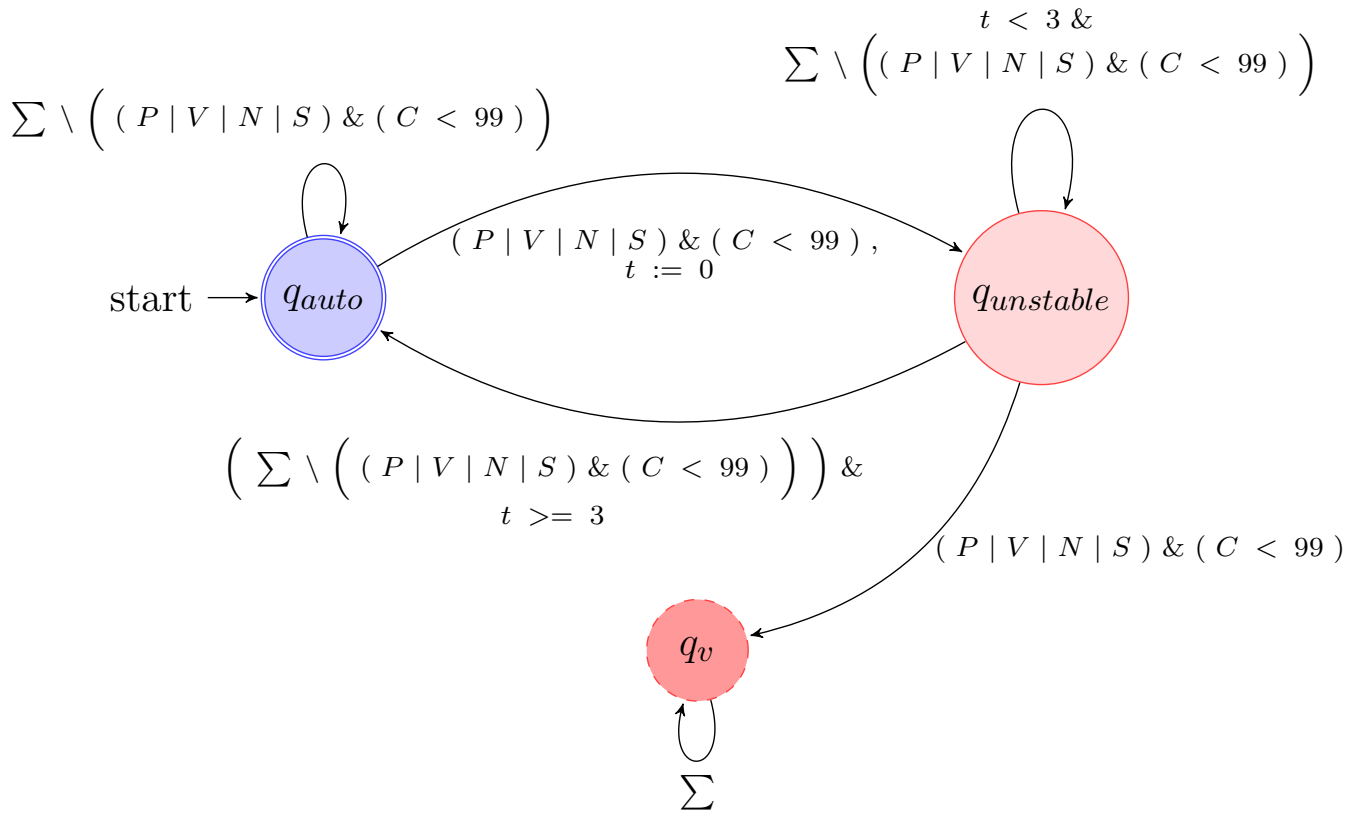

Figure 5.1: Block diagram showing the Meta Neural Network (MNN) ensemble

### 5.3.3 Input Perturbation Safety Policy

A safety (timed) automata **??** was designed to verify predictions made by the MNN and ensure utmost safety at all times. The automata started in a safe state, where control of the vehicle was autonomously handled by the system controller. If a misclassification was detected, the enforced policy entered an unstable state, still under autonomous control. Once enough time passed without further misclassifications, the vehicle entered the safe state again. However, if another misclassification was detected while unstable, the enforced policy entered a violation state and forced control of the AV to the driver. The vehicle would not enter autonomous mode again until the system was restarted. A diagram of the enforced policy's safety automaton is shown in Figure 5.2.

### 5.3.4 Runtime Enforcer for a Traffic Sign Recognition AV System

The system controller was encapsulated by a run-time enforcer [6] that used sensor fusion to check for misclassifications made by the MNN. This type of run-time enforcement, where neither the inputs nor outputs of the sensors or controller are enforced, has been termed as *run-time verification*. *Run-time verification* refers to the verification of system parameters during run-time, while ensuring that the system is aware of any failed guards in the enforced policy.




- $P$: Misclassification of a person.
- $V$: Misclassification of a vehicle.
- $N$: Classification of an object when there is nothing.
- $S$: Misclassification of a traffic sign.
- $C$: Confidence rating of the SSNN classification.
- $t$: Timer for the unstable state.

Figure 5.2: Enforcer policy for the Autonomous Vehicle (AV) prediction system

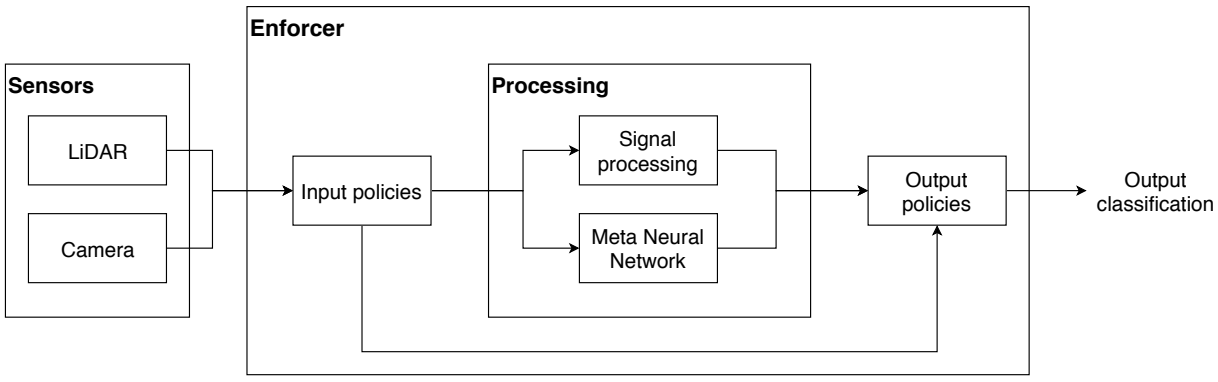



Figure 5.3: Block diagram showing the Autonomous Vehicle (AV) system with enforcer

## 5.4 Results of the Runtime Verified AV System

This research provides a solution for two aspects of Autonomous Vehicle (AV) systems: predicting accurately with perturbations to the system's inputs and safely dealing with misclassifications by the system. The issue of input perturbations was addressed using a Meta Neural Network (MNN) of different convolutional Synchronous Neural Networks (SNNs), each SNN working in tandem to predict more accurately. Misclassification by the system's controller was addressed by implementing sensor fusion between cameras and LiDAR. This was done using a run-time enforcer that enforced a safety automaton.

To test the MNN's ability to deal with perturbations, the input images (taken from the VOC 2012 and GTSRB datasets) were perturbed by randomly replacing approximately 7% of the image pixels with randomly coloured pixels. Table **??** shows that without perturbations, the accuracy of the MNN was increased from 87.07% to 93.7% when using an enforced policy. Table **??** shows that the accuracy of the MNN hugely decreases when perturbations are present, from 87.07% to 41.6%. However, with sensor fusion and a relative safety policy, the accuracy is increased to 90.48%.

| Object classified | No. of misclassifications (%) | Caught misclassifications (%) | False negatives (%) | Total remaining misclassifications (%) |
|---|---|---|---|---|
| Person | 17.01 | 17.01 | 0.00 | 0.00 |
| Vehicle | 15.85 | 15.85 | 0.00 | 0.00 |
| Sign | 54.46 | 54.46 | 4.84 | 4.84 |
| Nothing | 7.83 | 7.83 | 0.00 | 0.00 |
| All | 95.16 | 95.16 | 4.84 | 4.84 |

Table 5.1: Table showing the worst results of the AV prediction SNN using the original images, trained for 0 epochs

| Object classified | No. of misclassifications (%) | Caught misclassifications (%) | False negatives (%) | Total remaining misclassifications (%) |
|---|---|---|---|---|
| Person | 1.00 | 0.21 | 0.53 | 1.32 |
| Vehicle | 6.26 | 5.63 | 0.44 | 1.07 |
| Sign | 2.62 | 0.44 | 1.02 | 3.20 |
| Nothing | 2.50 | 2.27 | 0.12 | 0.35 |
| All | 12.38 | 8.55 | 2.11 | 5.93 |

Table 5.2: Table showing results of the AV prediction SNN using the original images, trained for 10000 epochs

(TODO: Add results for multiple epochs) (TODO: Add table showing how MNN affects the prediction accuracy) (TODO: Train ANN to run in place of the enforcer (i.e. decision making) and enforce it. Compare to original enforced system.)

| Object classified | No. of misclassifications (%) | Caught misclassifications (%) | False negatives (%) | Total remaining misclassifications (%) |
|---|---|---|---|---|
| Person | 1.11 | 0.70 | 0.65 | 1.07 |
| Vehicle | 3.92 | 3.34 | 0.65 | 1.23 |
| Sign | 2.64 | 0.60 | 0.51 | 2.55 |
| Nothing | 2.92 | 2.69 | 0.23 | 0.46 |
| All | 10.59 | 7.32 | 2.04 | 5.31 |

Table 5.3: Table showing the best results of the AV prediction SNN using the original images, trained for 6000 epochs

| Object classified | No. of misclassifications (%) | Caught misclassifications (%) | False negatives (%) | Total remaining misclassifications (%) |
|---|---|---|---|---|
| Person | 17.01 | 17.01 | 0.00 | 0.00 |
| Vehicle | 15.85 | 15.85 | 0.00 | 0.00 |
| Sign | 54.46 | 54.46 | 4.84 | 4.84 |
| Nothing | 7.83 | 7.83 | 0.00 | 0.00 |
| All | 95.16 | 95.16 | 4.84 | 4.84 |

Table 5.4: Table showing the worst results of the AV prediction SNN using perturbed images, trained for 0 epochs

| Object classified | No. of misclassifications (%) | Caught misclassifications (%) | False negatives (%) | Total remaining misclassifications (%) |
|---|---|---|---|---|
| Person | 3.43 | 0.60 | 0.56 | 3.38 |
| Vehicle | 10.78 | 8.57 | 0.23 | 2.43 |
| Sign | 38.08 | 31.91 | 1.39 | 7.56 |
| Nothing | 5.61 | 4.80 | 0.09 | 0.90 |
| All | 57.89 | 45.89 | 2.27 | 14.28 |

Table 5.5: Table showing results of the AV prediction SNN using perturbed images, trained for 10000 epochs

| Object classified | No. of misclassifications (%) | Caught misclassifications (%) | False negatives (%) | Total remaining misclassifications (%) |
|---|---|---|---|---|
| Person | 3.22 | 1.02 | 0.79 | 2.99 |
| Vehicle | 9.20 | 7.25 | 0.53 | 2.48 |
| Sign | 33.30 | 27.65 | 2.71 | 8.37 |
| Nothing | 6.81 | 6.12 | 0.09 | 0.79 |
| All | 52.54 | 42.04 | 4.13 | 14.62 |

Table 5.6: Table showing the best results of the AV prediction SNN using perturbed images, trained for 700 epochs

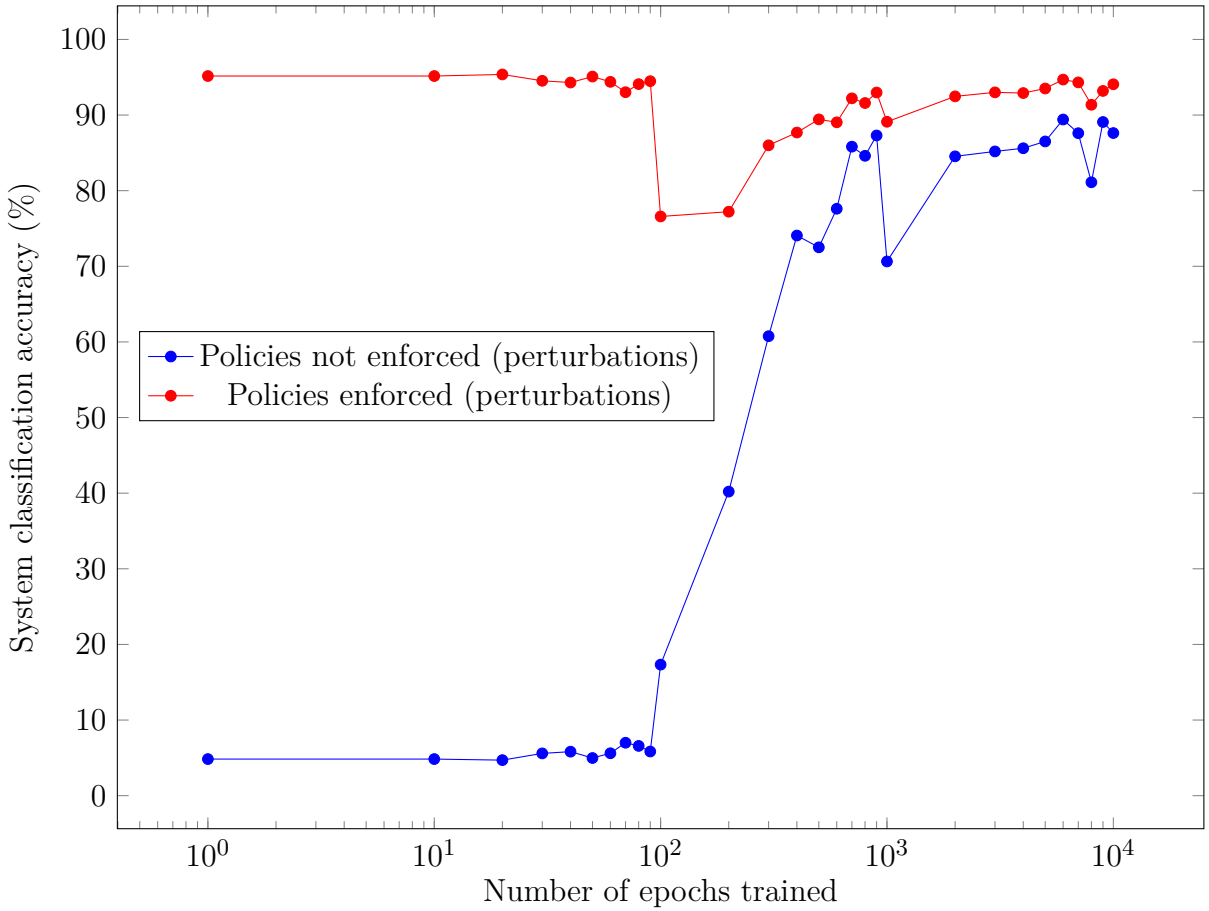



Figure 5.4: Line graph showing the performance of the system trained over an increasing amount of epochs using unperturbed inputs

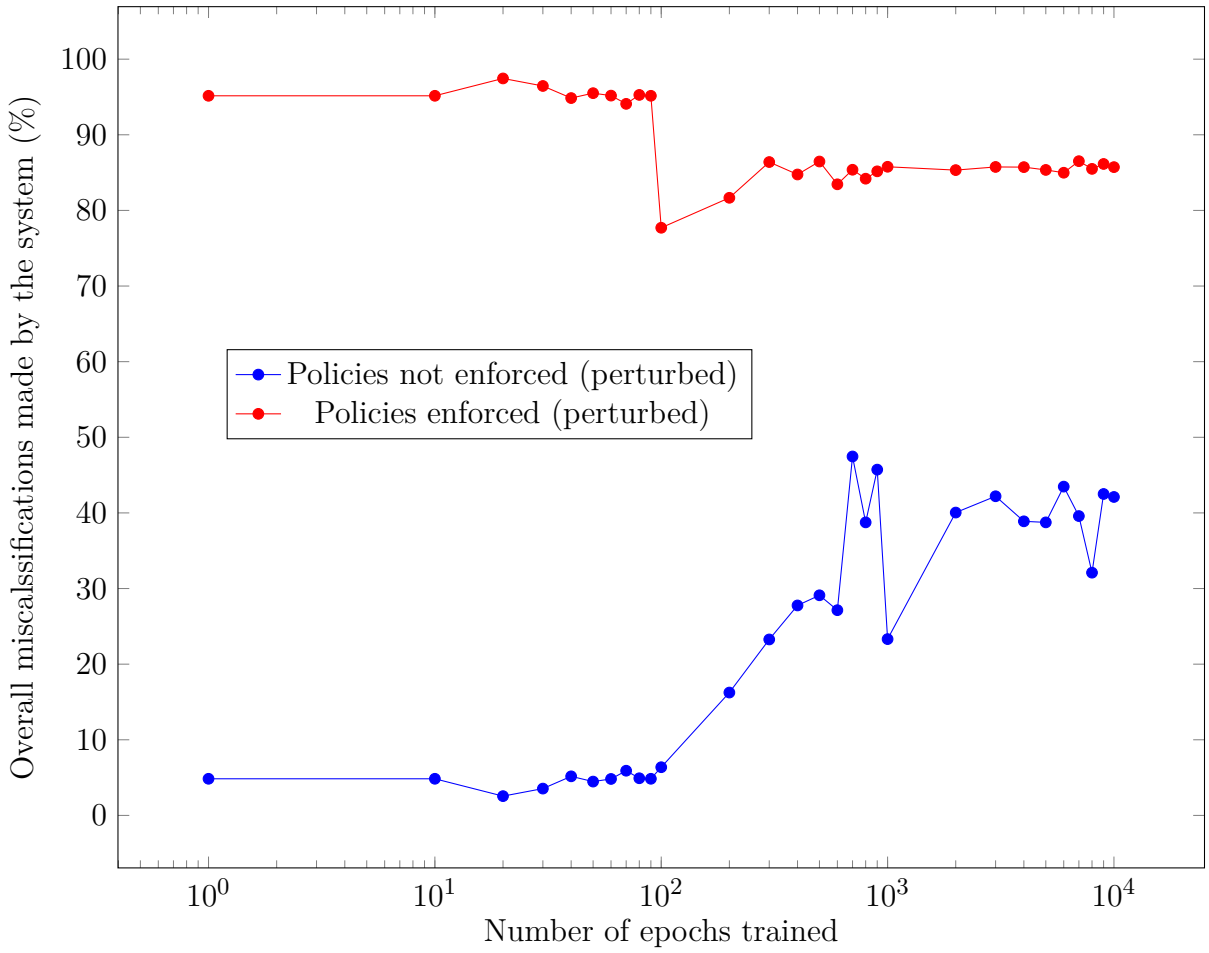



Figure 5.5: Line graph showing the number of misclassifications made by the system with perturbed inputs

## 5.5 Summary

In the previous chapter of this thesis, runtime enforcement was introduced to dynamically increase the safety of Artificial Neural Networks (ANNs). However, runtime enforcement is not always applicable to ANN systems; most notably where ANNs with complex inputs (such as Convolutional Neural Networks (CNNs)) are concerned. Thus, runtime enforcement is not a suitable solution to systems where CNNs are used, such as Autonomous Vehicles (AVs).

Inspiration for the safety of AVs was taken from the recent AV accidents by Uber and Tesla vehicles. These accidents resulted in the loss of life due to misclassifications made by the system controller, which consists of some layout of CNNs and other intelligent systems. Additionally, perturbations to a CNN's input can greatly decrease the prediction accuracy of a CNN. This, in turn, increases the chance of a misclassification by the system's controller and decreases the safety of the system.

In this chapter, two techniques are implemented to increase the safety of Convolutional Neural Networks (CNNs) in Autonomous Vehicle (AV) systems. There are two ways that a system that misclassifies data can be made safe: reduce the chance of misclassifications and act in a safe manner should a misclassification occur. An AV simulation was created to simulate an AV system as closely as was possible.

To address the first, a complex Meta Neural Network (MNN) was implemented as the system's controller. This MNN consisted of three different MNN ensembles, each ensemble classifying a different aspect of the detected object. Each MNN ensemble consisted of three different synchronous CNNs, with the outputs of each CNN combined to increase the prediction accuracy of the system. The three ensembles' outputs were then combined synchronously during runtime to further increase the prediction accuracy of the system controller. This MNN also served to increase the prediction accuracy of the system when perturbations were affecting the inputs to the MNN.

The simulated system also used sensor fusion, managed by a safety policy and enforced by a runtime enforcer. The system combined Light Detection and Ranging (LiDAR) input with the MNN, thereby increasing the chance of detecting a misclassification. Additionally, a timed automaton was used as the safety policy; this allowed the system to check for consecutive misclassifications and enforce the control of the vehicle where necessary.

The runtime enforcement used in this system hs been termed "runtime verification", as the outputs of the MNN itself were not enforced. Rather, the control of the vehicle (autonomous or driver operated) was enforced. If a situation was deemed unsafe by the safety policy, the runtime enforcer would force control of the vehicle to the driver.

This system was tested and trained using a large combination of images from the VOC and GTSRB datasets, with images of cars and people taken from the VOC dataset and

images of various traffic signs taken from the GTSRB dataset. The system showed that input perturbations can reduce the accuracy of the system by as much as 45% for a fully trained system. The efficacy of the implemented techniques are shown: both with and without the enforcer, the accuracy of the system controller greatly improves, with the enforced policy catching 45 out of every 57 misclassifications.

## 5.6 Discussion

This chapter presents techniques to increase the safety of AV systems. The first technique involves creating MNN ensembles, consisting of multiple CNNs, which work together to increase the prediction accuracy. The second technique uses runtime enforcement to verify the integrity of the MNN's classifications, attempting to detect misclassifications dynamically and put the vehicle in safe state if one is detected. These two techniques work in tandem to greatly increase the safety of the AV system where object misclassification is concerned.

Creating a MNN with three synchronous CNNs, forming a MNN ensemble, did have a large effect on the classification accuracy of the system. The classification accuracy was higher than that of an ordinary CNN regardless of the number of epochs trained, but it was noted that the increase in accuracy was smaller the better trained the ordinary CNN was. This means that using MNN ensembles is more effective for poorly or incompletely trained CNNs, but it still increases the prediction accuracy even when extensively trained CNNs are used. However, implementing three CNNs is very resource intensive; each has to be trained separately to a suitable standard and on different data, each has to be implemented separately and each has to run separately. This takes much more time, memory and processing power than implementing a single CNN to classify the input images. The MNN used more than three times the resources that a single CNN would use, but it can be argued that this trade-off is worth the increase in classification accuracy where human lives are concerned.

The second technique, the runtime enforcer, increased the prediction accuracy by 43% for an extensively trained MNN when input perturbations are involved. The system only improved by 6% when there were no input perturbations. The enforcer provided an increase in performance regardless of the amount of training the MNN had. With no training, the classification accuracy of the system matched the accuracy of the LiDAR sensor. As the system trained, and the MNN became more confident with its decisions, the accuracy of the system began to match the accuracy of the MNN. If the system were to be trained for more than 10,000 epochs, and the accuracy of the MNN exceeded the accuracy of the LiDAR, the overall accuracy of the system would likely increase beyond the accuracy of the LiDAR. More extensive training and testing of this system is proposed for future work.

The biggest downside to this system is that it uses three different MNNs to run. Each MNN is an ensemble of CNNs, as introduced previously. This means that using this layout increases the resource usage of the system by 300%, as opposed using a single MNN ensemble, or 900% compared to using a single CNN. However, the trade-off for the safety of the system is well worth the extra resource cost. Future work could look into

decreasing the resource intensity of this system.

# 6
# Conclusions

This is the conclusion

## 6.1 Future Work

# A

# Appendix

# References


[1] A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. Le Guernic, and R. De Simone, "The synchronous languages 12 years later," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 64–83, 2003.

[2] G. Berry and G. Gonthier, "The ESTEREL synchronous programming language: Design, semantics, implementation," *Sci. Comput. Program.*, vol. 19, no. 2, pp. 87–152, Nov. 1992.

[3] D. Coldewey, "Uber in fatal crash detected pedestrian but had emergency braking disabled," *TechCrunch*, May 2018. [Online]. Available: https://techcrunch.com/2018/05/24/uber-in-fatal-crash-detected-pedestrian-but-had-emergency-braking-disabled/

[4] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results," http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[5] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. T. Vechev, "Ai2: Safety and robustness certification of neural networks with abstract interpretation," *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 3–18, 2018.

[6] S. Pinisetty, P. Roop, S. Smyth, N. Allen, S. Tripakis, and R. von Hanxleden, "Runtime enforcement of cyber-physical systems," vol. 16, pp. 1–25, 09 2017.

[7] P. S. Roop, H. Pearce, and K. Monadjem, "Synchronous neural networks for cyber-physical systems," in *MEMOCODE '18 Proceedings of the 16th ACM-IEEE International Conference on Formal Methods and Models for System Design*, 2018.

[8] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[9] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural*

*Networks*, no. 0, pp. –, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608012000457

[10] P. Wei, L. Cagle, T. Reza, J. Ball, and J. Gafford, "LiDAR and Camera Detection Fusion in a Real Time Industrial Multi-Sensor Collision Avoidance System," *ArXiv e-prints*, Jul. 2018.

[11] T. Wu, C. Tsai, and J. Guo, "Lidar/camera sensor fusion technology for pedestrian detection," in *2017 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, Dec 2017, pp. 1675–1678.