Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Patrick Moen Allport

Applications of fault-tolerant software architecture principles in the detection of adversarial attacks

Master's thesis in Master of Science in Informatics
Supervisor: Leonardo Montecchi
June 2023

NTNU
Norwegian University of
Science and Technology

Patrick Moen Allport

Applications of fault-tolerant software architecture principles in the detection of adversarial attacks

Master's thesis in Master of Science in Informatics
Supervisor: Leonardo Montecchi
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU

Kunnskap for en bedre verden

Applications of fault-tolerant software architecture principles in the detection of adversarial attacks

IT3920 - Master's Thesis for MSIT

Patrick Moen Allport

Supervisor:
Leonardo Montecchi

Department of Computer Science
Faculty of Information Technology and Electrical Engineering
Norwegian University of Science and Technology
Spring 2023

Abstract

In November of 2022, the European Union Agency for Cybersecurity (ENISA) released its 2022 ENISA Threat Landscape Report (ETL), which describes the observed threats in the cyber domain between July 2021 and July 2022. In this report, they note the threat posed by adversarial attacks and described it as '... growing and represent a major threat in ML or AI domains.'.

Noting this threat, a Systematic Literature Review (SLR) was previously performed in order to understand the research field of detecting these adversarial attacks. Multiple papers have attempted to mitigate the threat posed by producing detection models which detect these adversarial attacks. A small minority of these papers used fault-tolerant principles and architectures to enhance the capabilities of their detectors. However, the SLR noted that these papers only discussed architectures specific to their implementation and did not explore the general concept of utilizing fault-tolerant architectures for detection. This thesis wishes to explore this research gap, by approaching the problem of detecting adversarial attacks from a fault-tolerance perspective.

Based on a customized version of an autonomous driving dataset, nuScenes, an adversarial dataset was generated. Using FGSM, Carlini&WagnerL2, APGD, and Shadow Attack, 60 000 adversarial examples were generated. Two fault-tolerant architectures were implemented and trained to detect these attacks. The first was the recovery block, consisting of an implementation of 'InputMFS' from Paula Harder et al. and a transfer-learned ResNet-101 model. This architecture would explore how multi-model architectures would detect adversarial attacks. The second fault-tolerant architecture was N-Version Programming (NVP). Utilizing multiple transformations, shown to improve detection, this architecture would explore how diversity in data would affect detection performance.

The recovery block results show that sequential detection is viable, but requires disjunct sets of detected attacks and is prone to incorrect configurations of architectures. The NVP architecture shows that increases in data can provide improved performance, but optimal compositions of data can be difficult to discern. Additionally, the composition of data can determine the detection performance across attacks. Both architectures show that fault-tolerant principles are viable, but introduce their own set of considerations. As a result, future works should explore methods to more efficiently determine optimal configurations and compositions of both component detectors and data.

Acknowledgement

I would like to thank my supervisor Leonardo Montecchi for his invaluable support in producing this work. Your ability to approach, identify and solve problems has been an inspiration. Working with you for the past year has given me the tools to understand the scientific field and the grit required to be an active member of it. In stressful and hectic times, you have been able to provide insights and counsel which have brought attention to what is important. This thesis would not be possible without your guidance and I wish you all the best in all your future endeavours.

- Patrick Moen Allport

Contents

Chapter 1

Introduction

In November of 2022, the European Union Agency for Cybersecurity (ENISA) released its 2022 ENISA Threat Landscape Report (ETL) [1]. The report details the threats observed in the cybersecurity domain between July 2021 and July 2022. The report lists 8 *prime* threats, which they regard as the most alarming in the cybersecurity domain. Number four on this list is *Threats against data*. In the chapter discussing such threats against data, they note that:

> «On top of this, Machine Learning (ML) and Artificial Intelligence (AI) is increasingly being adopted and is boosting the migration from traditional software systems based on deterministic algorithms to systems where ML or AI models use reason on data to calculate a solution for individual instances of a problem. This migration poses a new wave of risks that push toward the 'AI Act', a proposed European law on artificial intelligence (AI) – the first law on AI by a major regulator anywhere. »[1, p. 63]

The chapter on threats against data goes on further to tie some of these threats to machine-learning models by saying :

> «In addition to data leaks and data breaches, the increasing adoption of ML or AI models at the core of novel distributed systems and decision-making put data manipulation under the spotlight. Data poisoning and adversarial attacks become widespread with the aim of undermining trust in IT and production systems and, more generally, in society as a whole.»[1, p. 63]

In addition to repeatedly ranking *threats against data* highly in their reports [1, p. 63], they also note the risks tied to manipulating machine learning models, describing these attacks as '... growing and represent a major threat in ML or AI domains.'[1, p.67].

The realization of this threat is also not lost on the commercial space, with Google announcing their 'Unrestricted Adversarial Examples Challenge' in 2018 [2], as well as Microsoft releasing their 'Adversarial ML Threat Matrix'. The matrix was made in collaboration with the non-profit MITRE and companies including IBM and NVIDIA, and also with input from researchers at the University of Toronto, Cardiff University, and the Software Engineering Institute at Carnegie Mellon University [3].

The Adversarial ML Threat Matrix defines several security threats to machine learning models. Defining these threats as *adversarial attacks*, the matrix subdivides these attacks based on what the attack is trying to achieve. A subset of these adversarial attacks, called *evasive* adversarial attacks, are defined by the attack's objective of trying to prevent or disrupt the proper classification of an object in an image.

Previously, a systematic literature review (SLR) was executed by the researcher, focusing on understanding the research field of detecting *evasive* adversarial attacks [4]. The literature noted that the field had different approaches to detecting evasive adversarial attacks. Most of the experiments performed some data manipulation before passing them on to a trained detector. Quite a few of these showed strong performance, such as Chen et al. [5]. However, Chen et al. also noted that if an attacker knew the structure of their detection model, the attacks could be adjusted to degrade their detector's performance significantly.

A subset of papers approached the problem differently by implementing fault-tolerant techniques, often seen in software architecture systems. An example is utilizing multiple models, each optimized for detecting a specific set of attacks. Fan et al. [6] is an example of this, where the utilization of two detectors selected for covering each other's weaknesses increased accuracy with a nominal increase in false positives. However, implementing these multi-stage detectors was often strictly implementation specific and never systematically explored the use of fault-tolerant software architecture principles.

1.1 Research Questions

With the insights from the SLR, this thesis will attempt to fill a research gap by exploring how implementing fault-tolerant architectures can improve the performance of adversarial attack detectors. The research questions were written to explore more generalizable fault-tolerance concepts and how they could translate into more effective implementations of adversarial attack detectors. Thus, the following research questions were set for this thesis:

- **RQ1**: How can fault-tolerant architecture principles be applied to detecting adversarial attacks?
  - **RQ1.1**: How can multi-model architectures be used to improve the detection of adversarial attacks?
  - **RQ1.2**: How can diverse data be used in fault-tolerant architectures to improve the detection of adversarial attacks?

1.2 Thesis Outline

Excluding the introduction and conclusion, the thesis consists of 5 sections. The ‘background’ section explains the theory behind computer vision and how adversarial attacks are generated. The section further presents the taxonomy and principles of fault tolerance and how they tie into fault-tolerant architectures. The ‘thesis objective’-section explains the results and conclusions from the SLR and presents the scope and objectives of the thesis. The ‘method’ section presents an overview of the prerequisite artifacts required for the thesis and the tools used to implement them. The section describes how these artifacts were produced and the decisions made during their production. The results section describes the results found based on the artifacts and implementations from the method section. Finally, the discussion section ties the results to fault tolerance and what insights it provides regarding the research questions posed.

The complete codebase, datasets, and weights are available in the delivery system for this thesis. However, the weights and datasets are large, with the codebase being 36GB when compressed. Due to the delivery system’s limitations, it was impossible to deliver the code and data separately. To allow an initial viewing of the code without having to download everything right away, a GitHub repository containing only the code is available at: https://github.com/pmAllport/IT3920-delivery

Chapter 2

Background

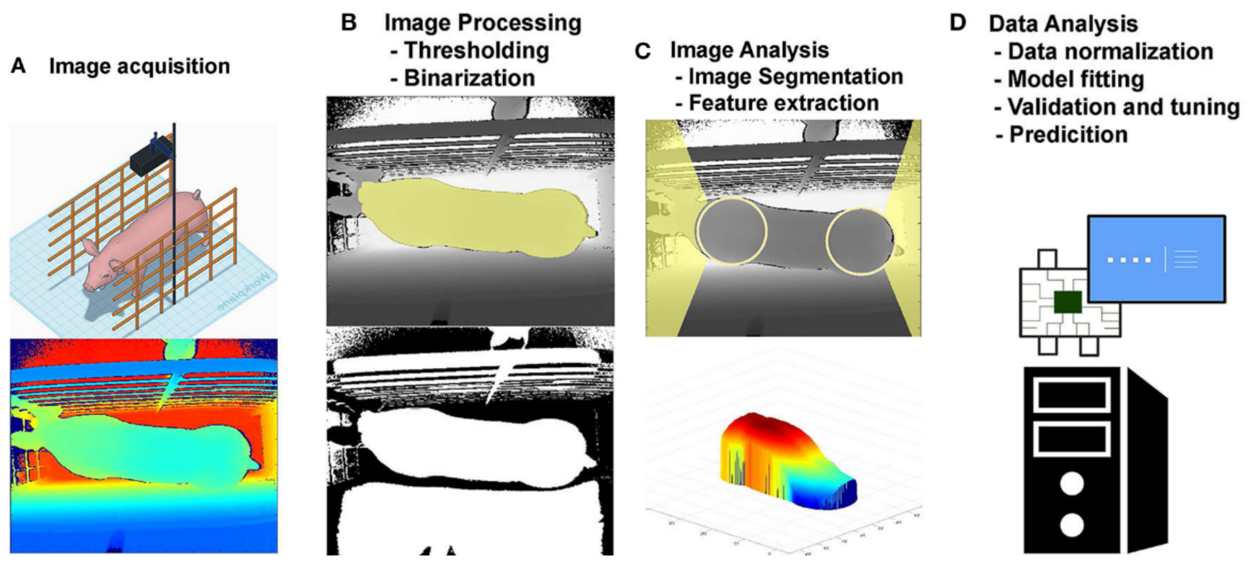
In order to understand the process presented in the 'method' section and the produced results, it is vital to explain the background on which they are based. This section will explain the concept of computer vision, how adversarial attacks function, and how fault-tolerance concepts are applied in architectures.

2.1 Computer vision

Computer vision describes the operations in which a raw image is converted from photons hitting a sensor to outputting usable data [7]. This is often done in multiple stages, each performing some operation to transform the image data. These stages may perform task-agnostic enhancements, such as image sharpening or noise removal, or task-specific enhancements, such as object segmentation. This section will present the stages of computer vision and how these stages produce information optimized for computer-vision models to utilize.

According to Victor et al. [7], computer vision aims to generate data for *Image understanding.* When passed to a computer-vision model, this 'understanding' can then be used by the system to make decisions [7] [8]. However, these decisions are only as accurate as the data itself. Thus, a core principle of the computer-vision pipeline is to enhance robust and distinct features in the data while filtering out irrelevant noise [9].

Presented in Figure 2.1 is an overview of the computer-vision pipeline and can *generally* be divided into four stages: *image acquisition*, *image processing*, *image analysis* and *data analysis* [10]. *image acquisition* is an important pipeline stage, as it converts photons to analog data. However, as its processes fall outside the potential attack surface for adversarial attacks, they will not be discussed in detail.



(a) (b) (c) (d)

Figure 2.1: An overview of the computer vision pipeline.
Adapted from: A. F. Fernandes et al. [10]

Image processing is the operation in which one converts analog data from one or more sensors into a digital image [7][8]. In addition to the conversion to digital data, several enhancement techniques are applied. Generally, these enhancements are task-agnostic and enhance the quality of the image itself. Higher quality data can allow the later stages of the pipeline to extract more distinct features and thus result in better predictions. Examples of these image processing techniques include image sharpening and contrast adjustment [10][11][12].

The image analysis enhancements are more task-specific as they are (generally) more strongly coupled with the task of the final model. What image enhancements are applied vary based on the input's attributes and what robust features the model requires. Information may be combined, filtered, mapped, or otherwise transformed to produce relevant data. The result may be visual, such as color maps or object segmentation, but may also be meta-data, such as histograms or the mean L2 distances in the image [12]. What features to extract can be set by the implementer and is often based on what they deem useful in generating image understanding.

This final stage is data analysis. This stage trains a model to perform predictions based on the features. While the implementations vary, some processes are typical in the data analysis stage. Generally, the features are passed to an implementation-specific model, which has been trained to provide correct classification based on said features. The processes performed depend upon the model architecture but may consist of convolutional, batch normalization, and dropout layers. Finally, it is passed to a fully connected layer producing logits and possibly a squashing function (e.g. sigmoid) to yield an output describing the classifier's prediction of one or more objects. The output can be described as an instance of *image understanding*, as it describes some property of the image provided. This image understanding 'data' can then be passed to a decision-making system for further use.

A thing to note on the presented computer vision pipeline is that the individual stages and the location of operations may differ between implementations. Some models may be responsible for, e.g. prepossessing, extraction of features, or selection of features [13]. The pipeline presented resembles the architecture of a traditional machine-learning computer vision pipeline. This is mainly done for ease of understanding.

This distinction is further complicated by the existence of both single-stage and dual-stage classifiers [14]. Single-stage classifiers, such as 'You Only Look Once' (YOLO), perform object detection and classification in a single pass [15]. Taking YOLO as an example of a single-stage detector, these detectors treat detection as a regression problem by iteratively defining, subsetting, and merging bounding boxes which give the highest classification probability for an object [16]. Dual-stage detectors, on the other hand, perform two passes, where the initial stage performs object segmentation and passes it on to the second stage, which performs classification as well as fine-tuning the bounding boxes [14]. In short, the computer vision pipeline presented is not strictly applicable to every implementation but is rather used to describe the processes which need to be performed overall to produce *image understanding*.

2.2 Adversarial attacks

Within the field of computer vision, machine-learning models are heavily used. They provide the ability to accurately perform object detection and classification, even in complex scenes. However, these models are also known to have vulnerabilities that are difficult to mitigate [17]. The abuse of these vulnerabilities is called an adversarial attack. The goal of adversarial attacks is to introduce an input, called adversarial examples, which coerces a model into an unintended state [18].

As mentioned in chapter 1, these attacks can be categorized based on the attacker's goal. A few notable mentions include *model extraction attacks*, which attempt to extract information about the model or *poisoning attacks*, which attempt to induce unexpected behavior (*trojans*) into the model [19]. Additionally, one has *evasive* adversarial attacks, which attempts to hide, prevent, or otherwise disturb the proper classification of objects in the scene. A visual example of *evasive* adversarial attacks can be seen in Figure 2.2, where the introduction of a small amount of noise can significantly affect a classifier. The figure shows how introducing noise causes a classifier to mistake a panda for a gibbon. It is these kinds of attacks that this thesis will look into. As a result, it is important to note that for ease of reading, any reference to adversarial attacks should implicitly be understood to mean *evasive* adversarial attacks.

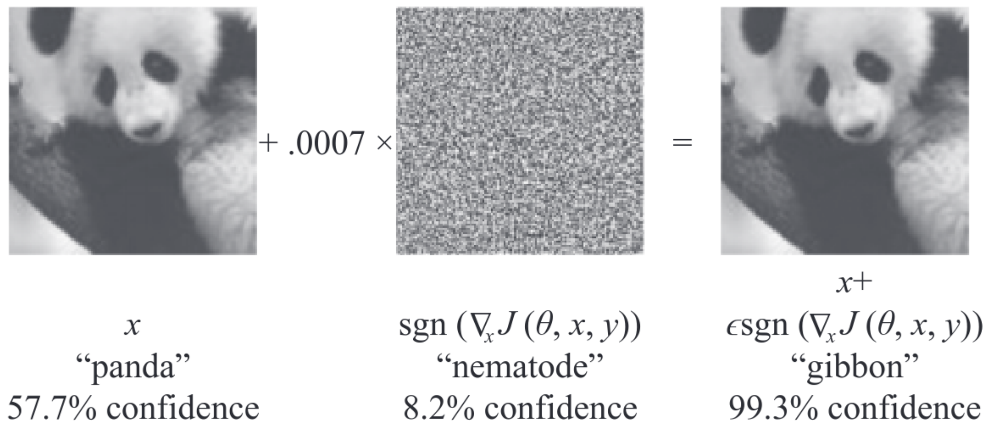


Figure 2.2: A figure showing an initial image, the noise applied, and the resulting image. The classification for each image is also shown below each image. Image credit goes to Goodfellow et al. [17]

Adversarial attacks generally treat the generation of adversarial examples as an optimization problem, attempting to determine some noise in an image that maximizes the model's loss if a given classifier were to attempt classification on said image [20] [21]. These attacks achieve this by utilizing a classifier's *gradients* to calculate this noise. These *gradients* represent the internal state of the classifier and how it prioritizes different data points in an input. By knowing this internal state, an attacker can find an image that would trick the classifier to the largest degree.

These attacks can be divided into white-box attacks and black-box attacks, where the former has access to the model gradients while the latter does not. What generally differentiates them is that black-box attacks must perform additional processes to calculate an 'estimated' set of gradients instead of being provided with the actual model gradients [19]. This means that black box attacks, in addition to having to produce an adversarial example, also need to have a method for estimating the internal gradients of the attacked model.

Utilizing these gradients, either estimated or real, the adversarial attacks use a number of different strategies to determine the optimal noise to maximize loss. Fast Gradient Sign Method (FGSM) is a simple white-box attack used to determine this noise and exemplifies the vulnerabilities posed by machine-learning models [17]. FGSM introduces an imperceptible noise level in the areas where the weights are most valued. The noise introduced can scale relative to the dimensionality of the network and the magnitude of each weight vector. This allows a number of minor perturbations in an image to scale up to a sizeable collective shift, given that the machine-learning model has a sufficient number of layers. This process can be easily understood by viewing FGSM's optimization function:

$$\eta = \epsilon sign(\nabla_x J(\theta, x, y))$$

The goal of the function is to find the noise η, given the input image x, the ground truth label of the image y, the model θ, the scaling factor ϵ the max-norm function *sign* and the loss function of the model J. In essence, it attempts to find the noise which introduces the maximum loss, which is then scaled by a factor of ϵ. A multitude of algorithms exist which improve upon FGSM, like BIM [22] or PGD [23], but under the hood still apply the same principle as FGSM. There are also other adversarial attacks which calculate this loss entirely differently, some of which will be introduced in section 4.5 .

The evolution of adversarial attacks has not gone unnoticed, and several approaches have been implemented to mitigate the effects of adversarial examples. These countermeasures (also called 'adversarial defenses') can (generally) be divided into three categories: 'gradient masking', 'adversarial training' and 'adversarial detection' [24]. *Gradient masking*-based defenses base themselves on the fact that adversarial attacks need access to a representative set of gradients for the model's internal state. Thus, these defenses attempt to mask these gradients or make them difficult to estimate. An implementation of such a defense is 'Defensive Destillation' [20], whereupon an attempt is made to smooth out the gradients inside the classification model. Papernot et al. reasoned that the effectiveness of adversarial attacks could be mitigated by introducing data produced by a ML model into the training set of another model. This would result in a model with smoother gradients that are more difficult to estimate and perturb by an adversary. However, gradient masking-based defenses do not necessarily stop the existence of adversarial attacks. Rather, these defenses only make it more difficult to execute them, and methods capable of circumventing these defenses are known [21].

Adversarial training is the introduction of perturbed images into the training set to reduce the susceptibility of a model to adversarial examples [25]. The introduction of perturbed images is intended to train the model's weights to show indifference to adversarial perturbations and classify objects based on more robust image features instead. Sadly, adversarial training can have an adverse effect on the model's ability to classify benign images correctly, and Carlini et al. have shown that defenses based on adversarial training can also be circumvented [26]. The last group is *adversarial detection.* It consists of the attempt to detect adversarial attacks as they are presented to the model. In section 3.1, this thesis will go into more detail in terms of how these detectors function, as well as the different approaches used within the research field.

2.3 Fault-tolerance in software

As mentioned in chapter 1, the previously performed SLR noted a number of ways in which deep-learning models were used to detect adversarial attacks. Especially noteworthy were models that utilized techniques similar to fault-tolerance principles from software architecture design. Given that these models may safeguard critical systems' decision-making software, utilizing these techniques to detect adversarial attacks is not unreasonable. However, it is important to understand how and why these fault-tolerant techniques are generally implemented to understand their implications. This section will describe the taxonomy of dependability, show how it translates into fault-tolerance principles and present architectures which utilize fault-tolerant principles.

In this thesis, the software engineering taxonomy will mainly be based on 'Basic Concepts and Taxonomy of Dependable and Secure Computing' by Avizienis et al. [27] and chapter 14 of 'Handbook of Software Reliability Engineering' by McAllister et al. [28]. These works were selected due to their well-defined taxonomy and presence in the fault-tolerance field [29] [30].

2.3.1 Dependability taxonomy

Any piece of software has a defined objective, be it the processing of items online or controlling an autonomous vehicle. While they are both pieces of software, the environments in which these two pieces of software operate are completely different. One system must be able to potentially handle millions of orders at once, while the other must be able to detect and prevent potentially hazardous traffic situations. The difference in environments and functionalities translates to different risks. These risks are determined by many different requirements set upon the system. In addition to functional requirements, which dictate **what** a system should be capable of doing, there are also non-functional requirements. These non-functional requirements specify criteria in which to evaluate how well a system achieves a certain operational attribute.

One top-level attribute is 'Dependability', which describes the 'trustworthiness of a computer system such that reliance can be justifiably placed on the service it delivers' [31]. Dependability encompasses multiple *primary attributes*, such as reliability, and captures the ability to rely on a system to function correctly [27]. Within the dependability scope, reliability describes a system's ability to continuously provide a correct service within the dependability scope. Also within this scope is the term 'robustness', which describes a system's 'dependability with respect to external faults' [27]. Expanding on the term robustness, the attribute describes how the system's dependability is affected when exposed to a specific set of external circumstances. In terms of adversarial attacks, it represents the system's ability to provide a correct service when it is being attacked by adversarial examples.

Avizienis et al. [27] define three terms to explain the threats to *dependability*: *faults*, *errors* and *service failures* (often shortened down to 'failures'). A (service) failure is defined as the deviation between what a service is determined to provide and what it actually provides. An error is the partition of the system state, which can create or has already created this service failure. A fault is 'the adjudged or hypothesized cause of an error' [27]. It is important to note that these faults can be both internal or external and that external faults rely on internal faults to exist in order to affect the system [27]. Faults are further subdivided into *dormant* and *active* faults based on whether the fault is presently affecting the system's output.

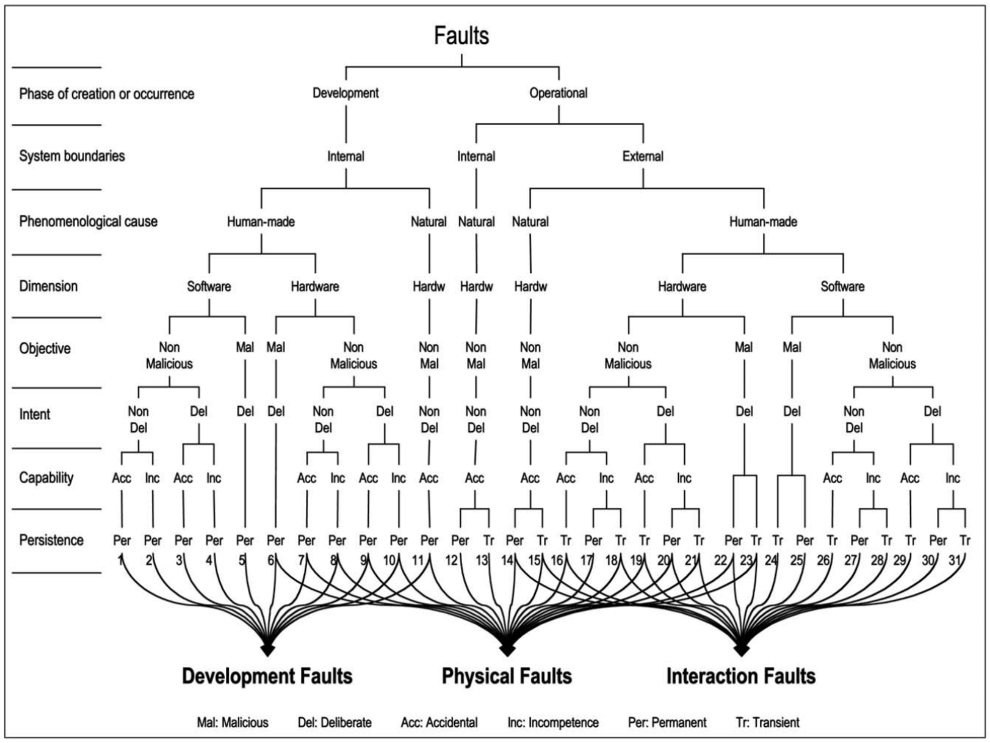


Figure 2.3: A figure of the tree-representation of faults from Avizienis et al. [27, fig. 5 b)]

Due to the robustness' connection to specific circumstances, only a subset of these faults are relevant when evaluating robustness for models being attacked by adversarial examples. Figure 2.3 shows an overview of the different classifications of faults. The issue of adversarial attacks can be classified as operational, external, human-made, software-based, malicious, and transient (classification 24). This is because the model is being attacked at run-time (operational), by an external influence (external), by a human-made attack (human-made), on a software level (software), by a malicious and deliberate actor (malicious/deliberate), and the attack only affecting the system when it is being attacked (transient).

Discussing adversarial attacks from a fault perspective, adversarial examples triggers a dormant fault which may manifest itself as changes to the output of some perceptrons in the neural net. This would be an example of an *error*. This *error* may or may not propagate throughout the neural net and ultimately cause the model to misclassify an object. If it does misclassify an object, the adversarial attacks has induced a *service failure* of the model. The model's inability to provide a correct service (i.e. 'correct classification') could then create *errors* (and even failures) in other 'downstream' systems and cause errors in those systems as well. This concept is defined by Avizienis et al. [27, sec 3.5] as 'error propagation' and McAllister et al. [28, chap 14.3.3] as 'dependent failures'. These terms do not overlap entirely but rather refer to the same concept of cascading errors and failures causing issues in downstream systems. Combining these two, one could say that 'error propagation' can cause 'dependent failures'.

Avizienis et al. [27] also define several mitigation strategies, which are grouped into four categories:

- Fault prevention, which attempts to prevent the initial creation of a fault.
- Fault tolerance, which is the ability to avoid service failures, given software faults.
- Fault removal, which is the process of reducing the number and severity of faults.
- Fault forecasting, which is the estimation of fault presence and the outcomes should the system fail.

As shown in section 2.2, some models can have their ability to provide correct service disrupted using adversarial attacks. This suggests that at least *some* models contain faults that a malicious actor can abuse. Some papers suggest that the existence of adversarial attacks is inherent to the structure of machine learning models [17] [32]. Others argue that accurate and stable neural networks are possible, but modern algorithms currently do not compute them [33]. One fundamental piece of knowledge in computer science is 'the halting problem', which proves that no general algorithm can show if an arbitrary computer program will halt or run forever, given a specific input [34]. As a result, it will be impossible to determine if a machine-learning model will fall victim to an adversarial attack. This is not to say that it is impossible to understand what creates these vulnerabilities, but rather highlights that the complexity of the problem makes it difficult to assertively state a cause for their existence.

Regardless of the cause of this vulnerability, the viability of adversarial attacks points to the fact that *some* current computer-vision models are likely to contain *some* level of dormant faults. These *dormant* faults can then be activated by an attacker via adversarial examples and induce an error in the attacked system. The existence of dormant faults in models makes mitigation via fault prevention a non-viable option, as the faults are already present in models. Fault removal as a strategy can be a valid strategy against trojaned models [35]. However, *evasive* adversarial examples do not require the model to be initially trojaned to disrupt the model. Fault forecasting is useful for designing the system and properly defining requirements for a system, but is in itself not helpful for preventing adversarial attacks. Thus, improving a model's performance against adversarial attacks is the only viable option to improve its fault tolerance.

2.3.2 Fault-tolerance principles

Treating the models as software components with known dormant fault problems enables the utilization of fault-tolerant principles. The utilization of such principles is referred to by Avizienis et al. and McAllister et al. as *fault tolerance* or *tolerance* [27, sec 3.4] [28, sec 14.3.5] respectively. Fault tolerance refers to a system's ability to detect, mitigate or otherwise prevent the inducement of an error given a fault. Avizienis et al. and McAllister et al. both note the importance of *design diversity* and *redundancy* as key to enhancing fault tolerance in a system.

Redundancy is the multiple computations of an input intended to avoid common-cause faults in two or more systems [28]. Architectures implementing redundancy-based solutions can be configured to perform these computations in parallel or sequentially. In the case of sequential redundancy, the need for additional computations can be determined based on some function. These functions are called *Acceptance Tests* (AT), and a common AT is seeing if a value is above a certain threshold.

Tied into redundancy is the requirement for *design diversity* of these systems [36] [28]. Design diversity refers to the variety between each of the computation systems. An increase in design diversity increases the total *coverage* of the fault-tolerant technique, where coverage refers to the space of inputs in which the technique is effective at tolerating faults [27]. Within the scope of adversarial attacks, coverage would refer to the total space of adversarial attacks the system can detect. Thus, by having multiple diverse redundant systems, one increases the total number of adversarial attacks one can detect. This is because one system may not provide coverage for a given attack, but other systems might be able to do so. The diversity of these systems is vital to their success, as largely overlapping coverage would provide very little overall value to the system. As described by McAllister et al. [28] :

> «The overall philosophy is to enhance the probability that the modules fail on *disjoint* subsets of the *input space* and thus have at any time at least one correctly functioning software component. »

To understand coverage, it is important to understand the term 'input space'. Chapter 5 of the 'Handbook of Software Reliability Engineering' explains that the term 'input space' refers to the set of all possible *input states*. [37]. An input *state* describes a set of input *variables* which can be input into a system, where input *variables* are all the different parameters of a system that may affect its operation [37]. A practical example of this would be an adversarial image. Each pixel in the image is an input *variable*, the entire image is an input *state*, and the complete set of all valid permutations of images is the input *space*.

Further, a *run* is the process of taking an input *state*, performing some *operation* on the input *state*. For any given operation, the set of valid input states for that operation is called the *domain*. A 'run type' is the set of runs with the same input *state*, but not necessarily the same operation applied. Bringing this back to a practical example, a *run* is the process of taking an adversarial image (input from domain) and applying some detection algorithm to it (operation). A figure showing these terms can be seen in Figure 2.4

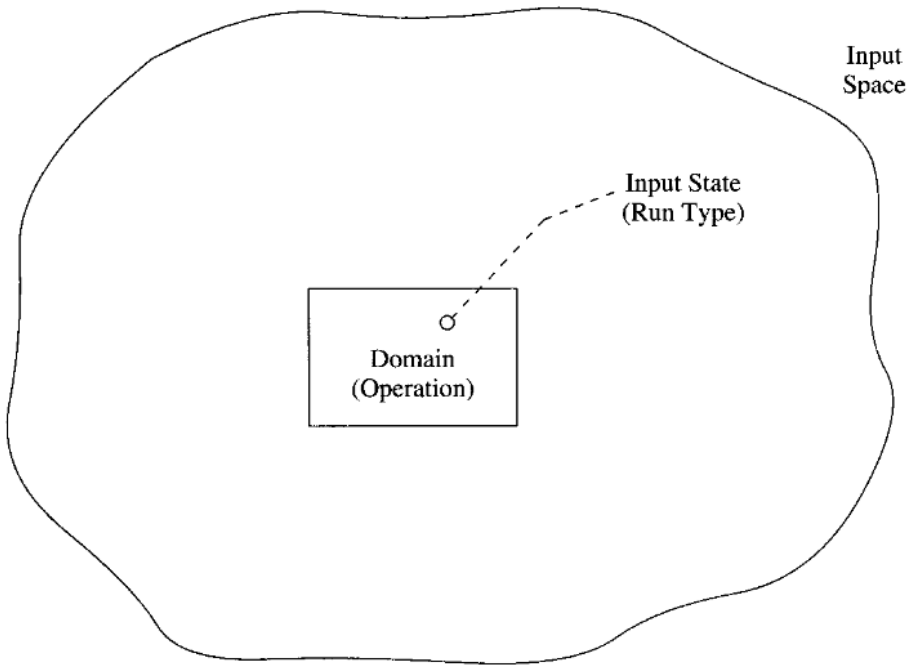


Figure 2.4: A diagram showing input space, input states, and domains.
Adapted from: J. Musa et al. [37] figure 5.1.

Understanding these terms allow us to describe how coverage is provided by a fault-tolerant technique. The level of coverage is dictated by the area covered by the *domains*. This means that to influence coverage, one can either change the input *space* or change the performance of the *operations*. In more practical terms, it means that the effectiveness of fault-tolerant techniques depends upon both data (input space) and detection methods (operations).

2.3.3 Fault-tolerance architectures

McAllister et al. [28] present several architectures that can increase a system's fault tolerance. Those relevant to this thesis include the *Recovery Block* (RB) and *N-Version Programming* (NVP) architectures. Figures of these models can be seen in Figure 2.5 and Figure 2.6.

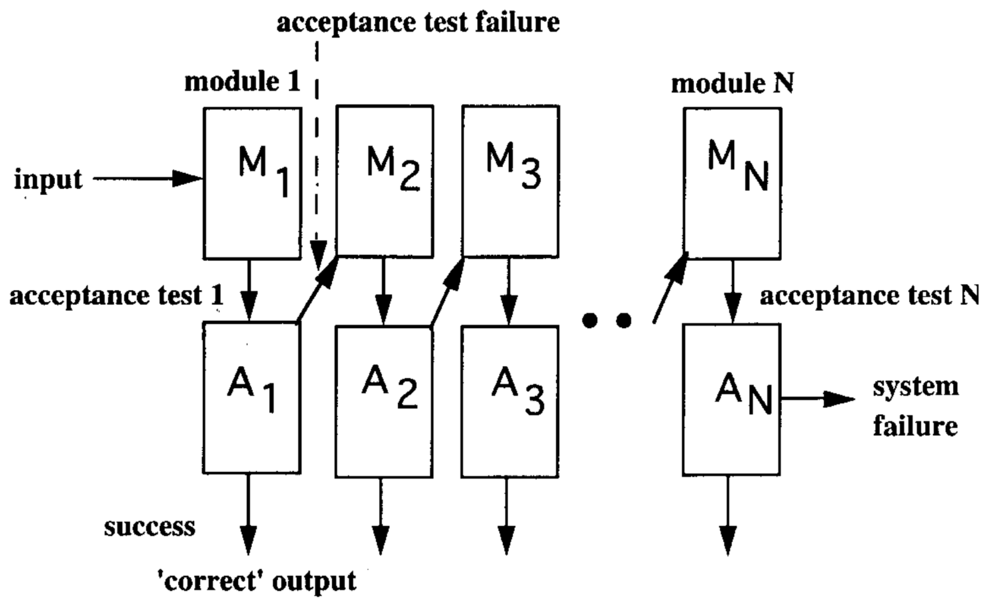


Figure 2.5: A schematic overview of recovery blocks from McAllister et al. [28, fig. 14.3]

Recovery blocks are built on two components: a module M and an acceptance test (AT) A. In a run-time scenario, the input is passed through the module M, producing an output passed to the acceptance test A. The acceptance test then makes a determination based on the output of M to adjudicate if the input passes or fails. If the input fails, the input may then be passed on to another set of modules and acceptance tests. This is repeated until an acceptance test determines the input to be acceptable or the input fails the final acceptance test. The complexity of the modules and acceptance tests may vary across implementations. McAllister et al. note, 'An extreme case of an acceptance test is another complete module, and the acceptance test would then consist of a comparison of a given module output with the one computed by the acceptance test. '.

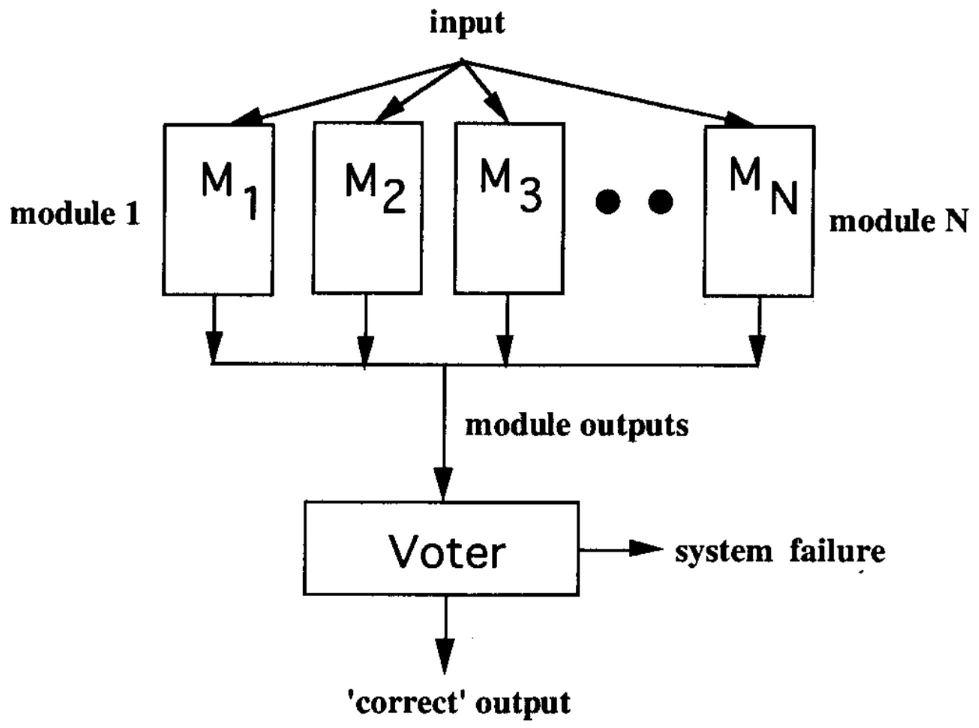


Figure 2.6: A schematic overview of N-Version programming from McAllister et al. [28, fig. 14.4]

N-Version Programming (NVP) is a software generalization of N-modular redundancy and involves the input being individually processed by multiple parallel modules, $M_0, M_1 ... M_n$. The output from the modules is then passed on to an adjudicator (also called a 'voter'), which takes these outputs as input and determines the best action based on these inputs. What operation the modules perform and how the voter makes its adjudication varies across implementations. Examples of how the voter makes its determination can be majority voting or median voting, but can generally be any function that bases itself on the output of the modules.

Chapter 3

Thesis Objective

The thesis topic and research field were set based on the insights from a previously performed Systematic Literature Review (SLR). In order to understand the rationale behind selecting this topic, a summary of this SLR must be provided. The first section will describe the insights the SLR generated and the research gap observed. The second section will explain how these insights tie into the fault-tolerance taxonomy and how they define the scope of this thesis.

3.1 Systematized literature review

In the fall of 2022, a Systematic Literature Review (SLR) was performed in the field of detecting adversarial attacks on computer vision. This SLR built the foundation for this thesis by giving insights into the field of detecting adversarial attacks. The thesis is largely based on the knowledge gained in this SLR, and its noted research gap forms the rationale for working on this subject. As a result, this section will present a summary of relevant works found in the SLR and the research gap uncovered in the work.

3.1.1 Findings

The findings from the systematic literature review were based on eleven different papers. These papers investigated and implemented different methods for detecting adversarial attacks. The SLR was aimed at understanding this research field and categorizing the current state of these detection methods. The SLR discovered that these detection methods could be categorized into two general groupings and synthesized two terms to define them: robustness-based methods and architecture-based methods.

3.1.1.1 Robustness-based methods

Robustness-based methods based themselves on the premise that introducing new transformations will disproportionately affect adversarial perturbations more than the features that were part of the original benign image. The *robustness* in 'Robustness-based' refers to the benign features' ability to resist change better than adversarial perturbations across image transformations. In the SLR, these robustness-based methods were further split into two categories based on how they perform their detection: consistency analysis and reconstruction deviation.

Consistency analysis performs its detection by understanding how some metrics in benign images change when introduced to one or more (generally linear) transformations. By knowing how these metrics should change, it can detect adversarial images if this image metric exceeds the bounds of what one could expect from a benign image. One example of this is Paula Harder et al. [38], where they perform a Discrete Fourier Transform (DFT) on an image and train a logistical regression model to discern between adversarial and benign images based on features from this Fourier spectrum.

Another example is Nathan Drenkow et al. [39]. In their paper, they perform random mutations on images and create a statistical model for how quickly the classification labels for benign images change as more mutations are applied. This statistical model utilizes additional factors, such as the consistency between the mutated images. Given an adversarial image, they assume that the labels will diverge at a different rate than the benign images. When this rate exceeds the bounds set by the statistical model, it will be detected as an adversarial image.

Reconstruction deviation generally describes methods that perform their detection using Generative Adversarial Networks (GAN) to regenerate an image. Using this regenerated image, a pre-trained detector is trained to learn how benign image features manifest in the regenerated images. An example of this is Wang, Shuo et al. [40], which takes an input image and encodes into the latent space for a given GAN. Their method then uses an optimal noise map to enhance the stochastic detail in the latent data as well as identify several style axes to shift their data. The images were then sent to a pre-trained detector which they used to differentiate between benign and adversarial images.

Another paper by Chen, Ruoxi et al. [5] performed their detection differently. Instead of working on the input image, they implanted their detector inside the classification model. Their method involves taking the features from the fully connected layer and training two GANs to generate the *salient* and *trivial* image features. The *salient* features denote the features which are more related to the class, while *trivial* features denote the features which provide very little in terms of classification. The maps containing these features are then passed to a detector, trained on where these maps are located on benign and adversarial images. If these maps do not correspond with what would be expected of a benign image, the input image is classified as adversarial. An example of their implementation is in Figure 3.1.

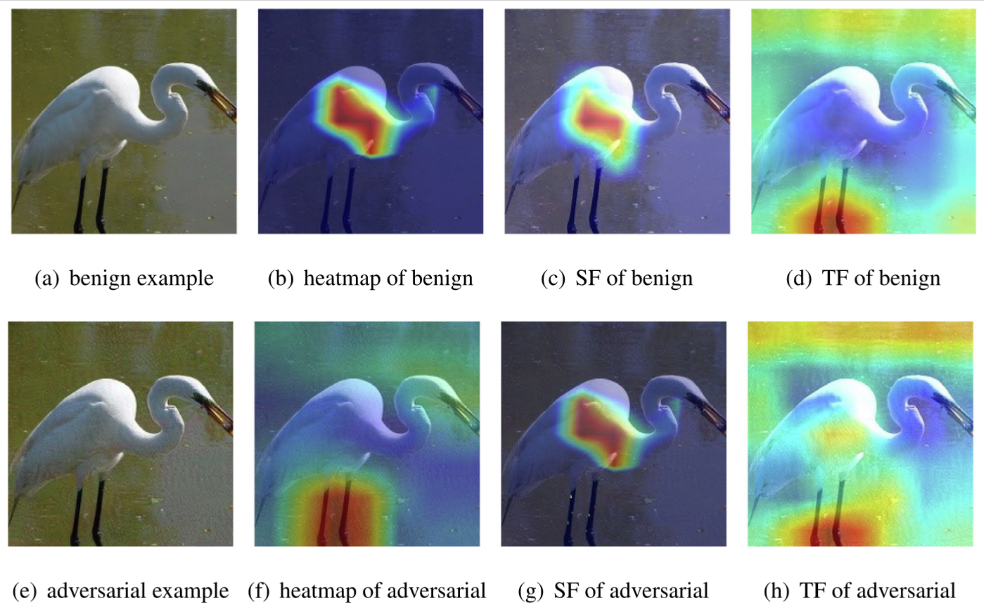


Figure 3.1: A set of 8 images showing examples of heatmaps of salient features (SF) and trivial features (TF) in adversarial examples.
Adapted from: Ruoxi, Chen et al. [5] figure 1.

3.1.1.2 Architecture-based methods

The other group of detectors described in the SLR are architecture-based detection methods. These methods were noted to rely on techniques often found in software architecture engineering, which try to increase the fault tolerance of a system. However, how this is achieved varies between the implementations.

An example of an architecture-based detector is Yang, Karren et al. [41]. They perform their detection by utilizing multiple sensors observing the same space, these sensors being video, audio, and LIDAR. By utilizing sensor fusion, they are able to identify if a sensor is being perturbed and use n-redundancy voting to out-vote the perturbed sensor. They base their work on the KITTI dataset [42], which consists of multiple eight-hour video feeds from cameras on a vehicle. Using such voting and sensor fusion strategies to detect adversarial attacks resulted in a mean average precision of 85%+ on images containing cyclists and cars. The utilization of this additional sensor data provides the detector with additional information in which to perform detection. This means that the increase in data provides an increase in coverage, as it is more capable of detecting an attack given a specific input.

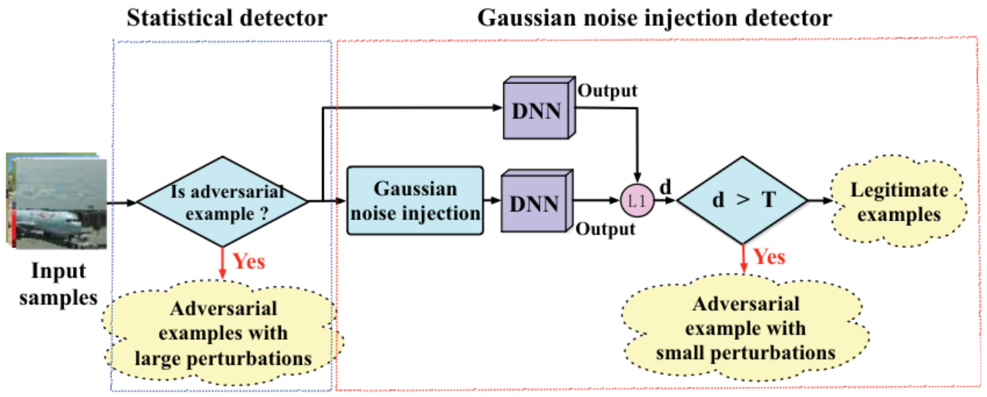


Figure 3.2: An overview of the 'hybrid detector', showing its multi-stage detection architecture.
Adapted from: Fan, Weiqi et al. [6] figure 3.

Another example of an architecture detector is the 'hybrid detector' by Fan, Weiqi et al. [6], with an overview shown in Figure 3.2. Their detection is done in two stages, the first being the use of a stenographic method called Subtractive Pixel Adjacency Matrix (SPAM). As they note in their paper, this method is adept at identifying large perturbations in an image. If this method detects that perturbations have been applied, the image is classified as adversarial. If classified as benign, the image is passed to the second stage, which produces a copy of the image injected with Gaussian noise. Both images are then passed to a GAN, which is trained to generate an output in which the adversarial perturbations will be more overt. The comparison of these two outputs is noted to be effective in detecting if small perturbations have been applied to the image.

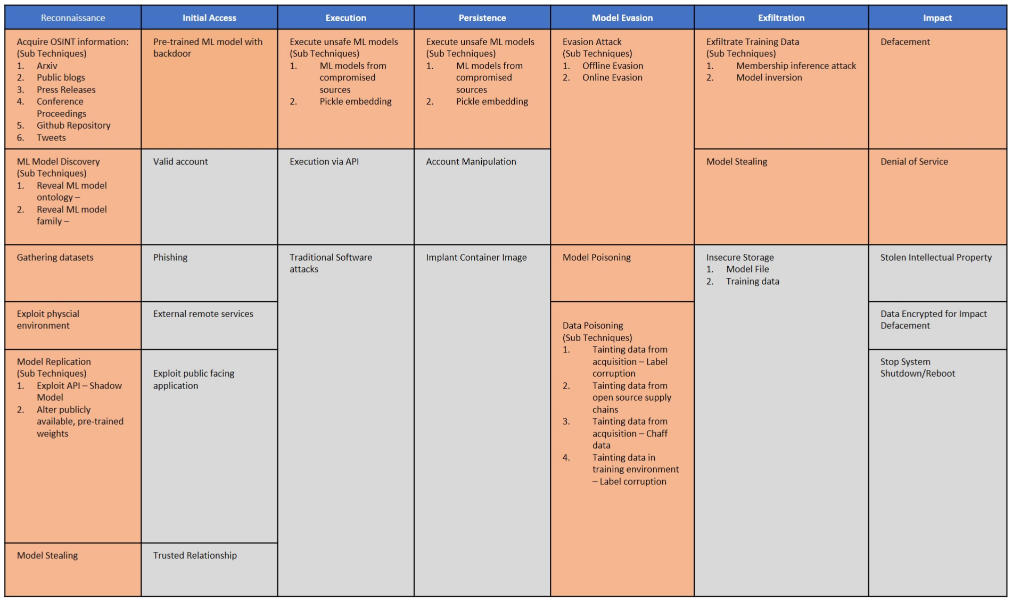
Comparing the 'hybrid detector' to the salient feature extractor by Chen, Ruoxi et al. [5] provides an insight into how these two groupings differ. Both detectors use a GAN-based approach, whereby they perform their detection based on a set of regenerated images. However, as noted in Chen, Ruoxi et al. [5], these GAN-based methods are vulnerable attacks to which large perturbations have been applied. When these large perturbations are applied, Chen, Ruoxi et al. notes that their detection rate falls from as high as 98%, down to 50-60% on the CIFAR-10 and ImageNet data sets. Fan, Weiqi et al. note that the first SPAM stage was added to mitigate this vulnerability to large perturbations [6]. Through the addition of this initial stage, Fan, Weiqi et al. are able to increase the accuracy by as much as 3% (96.2% to 99.5%) in their tests, with minimal cost to other metrics.

3.1.2 Conclusions

The resulting papers from the SLR showed a larger amount of papers focusing on improving individual models using robustness-based methods. Most of the implementations had a high level of accuracy but also came with a high level of false positives or neglected to provide that information. In general, the robustness-based methods performed better than their architecture counterparts. Still, as previously discussed when looking at Chen, Ruoxi et al. [5], the utilization of a single model can come with security risks. Contrasting the large presence of robustness-based papers, only a few papers attempted to approach the issue from a software architecture-based perspective. Here one sees novel approaches to detection, be it a simple filter to mitigate a known model weakness, as with Fan, Weiqi et al. [6], or introducing data points outside of the attack surface as with Yang, Karren et al. [41]. While it is difficult to directly compare the performance of architecture-based methods and robustness-based methods (due to different datasets and/or metrics), it can at least be observed that software architecture principles provide increased performance to the detection framework compared to its individual components. However, the architectures were implementation specific and did not discuss the general concept of architecture detectors.

3.2 Thesis scope

As briefly touched upon in section 2.2, the term Adversarial attacks covers a large collection of attacks. The ‘Adversarial ML Threat Matrix’ from Microsoft [3] shown in Figure 3.3, shows the sheer extent to which it is possible to disrupt the proper operation of a model using adversarial attacks. As a result, defining the boundaries of the thesis is key to ensuring that the thesis does not increase its scope outside what is possible, given the time frame and resources available.



Reconnaissance	Initial Access	Execution	Persistence	Model Evasion	Exfiltration	Impact
Acquire OSINT information: (Sub Techniques) 1. Arxiv 2. Public blogs 3. Press Releases 4. Conference Proceedings 5. Github Repository 6. Tweets	Pre-trained ML model with backdoor	Execute unsafe ML models (Sub Techniques) 1. ML models from compromised sources 2. Pickle embedding	Execute unsafe ML models (Sub Techniques) 1. ML models from compromised sources 2. Pickle embedding	Evasion Attack (Sub Techniques) 1. Offline Evasion 2. Online Evasion	Exfiltrate Training Data (Sub Techniques) 1. Membership inference attack 2. Model inversion	Defacement
ML Model Discovery (Sub Techniques) 1. Reveal ML model ontology – 2. Reveal ML model family –	Valid account	Execution via API	Account Manipulation		Model Stealing	Denial of Service
Gathering datasets	Phishing	Traditional Software attacks	Implant Container Image	Model Poisoning	Insecure Storage 1. Model File 2. Training data	Stolen Intellectual Property
Exploit physcial environment	External remote services			Data Poisoning (Sub Techniques) 1. Tainting data from acquisition – Label corruption 2. Tainting data from open source supply chains 3. Tainting data from acquisition – Chaff data 4. Tainting data in training environment – Label corruption		Data Encrypted for Impact Defacement
Model Replication (Sub Techniques) 1. Exploit API – Shadow Model 2. Alter publicly available, pre-trained weights	Exploit public facing application					Stop System Shutdown/Reboot
Model Stealing	Trusted Relationship					

Figure 3.3: Microsoft's Adversarial ML Threat matrix [3]. Machine learning attacks are shown in orange.
Used under license from The MITRE Corporation for research purposes [1]

As mentioned in section 3.1, the previous semester was spent investigating the different methods of detecting *evasive* adversarial attacks. Further, it also laid out the observed lack of generalized insights into the implementation of such architecture-based detectors. Observing the application of fault-tolerant principles for the architecture-based detectors, Fan, Weiqi et al. [6], and Yang, Karren et al. [41] both utilize redundancy and data diversity concepts. Fan, Weiqi et al. [6] utilize a sequential multi-model approach that iteratively removes disjunct sets of attacks. Karren et al. [41] utilize diverse data, which allows for the out-voting of attacked sensors and provides stronger coverage on a given input.

As noted multiple times, neither paper goes into significant detail in exploring how fault-tolerant architectures can be implemented in general. It is this research gap that this thesis wants to approach; how do some implementations of fault-tolerant architectures perform in detecting adversarial attacks, and what factors are likely to affect them.

This thesis intends to approach this issue of adversarial detection as a fault tolerance issue. This puts a premium on realistic scenarios as they can generally be expected to be noisier but also will produce results more likely to represent the environment in which these systems will run. Additionally, the fault-tolerant approach also reduces the importance of per-component performance. The thesis aims to understand how a fault-tolerant architecture's error detection compares to its constituent components. Thus, the thesis' goal is not necessarily to produce the most optimal components but rather to produce sufficiently rep-

[1]©2023 The MITRE Corporation. This work is reproduced and distributed with the permission of The MITRE Corporation.

resentative components and focus on differentiable approaches. This would then allow for a focus on what the architecture provides that each individual component cannot.

Treating the problem as a fault-tolerant system set the thesis up for utilizing fault-tolerant architectures. As mentioned in section 2.3, the coverage of a fault-tolerant technique was quantified in the context of 'domains'. These domains were defined by the input space and the operations applied. As a result, this thesis set itself up for the process of looking into how input spaces and multi-operation architecture models would affect the detection of adversarial attacks. It is this insight that dictated the research questions set in chapter 1.

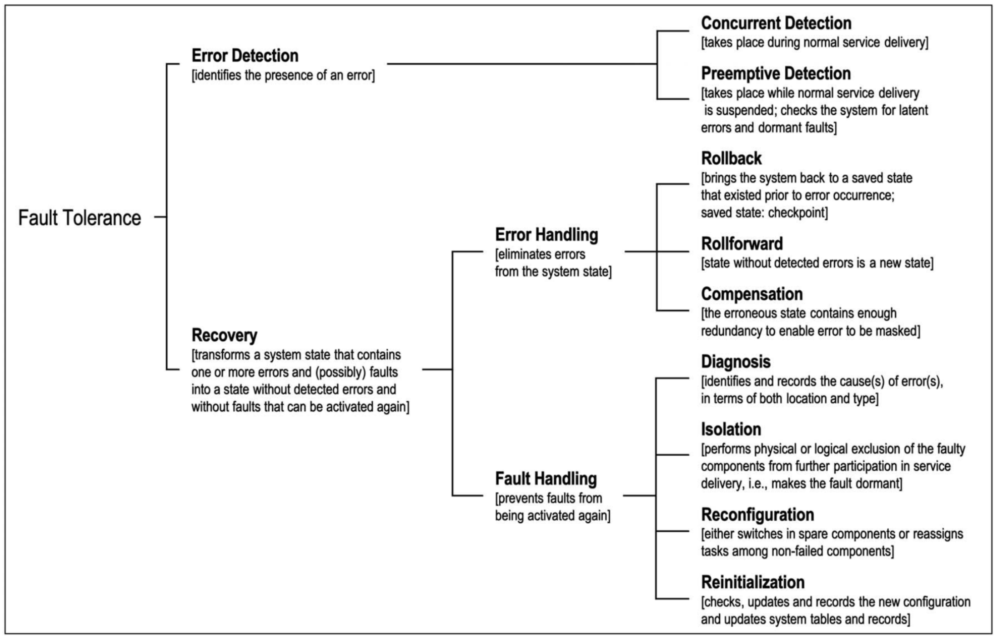

Fault Tolerance
Error Detection
[identifies the presence of an error]
Concurrent Detection
[takes place during normal service delivery]
Preemptive Detection
[takes place while normal service delivery is suspended; checks the system for latent errors and dormant faults]
Recovery
[transforms a system state that contains one or more errors and (possibly) faults into a state without detected errors and without faults that can be activated again]
Error Handling
[eliminates errors from the system state]
Rollback
[brings the system back to a saved state that existed prior to error occurrence; saved state: checkpoint]
Rollforward
[state without detected errors is a new state]
Compensation
[the erroneous state contains enough redundancy to enable error to be masked]
Fault Handling
[prevents faults from being activated again]
Diagnosis
[identifies and records the cause(s) of error(s), in terms of both location and type]
Isolation
[performs physical or logical exclusion of the faulty components from further participation in service delivery, i.e., makes the fault dormant]
Reconfiguration
[either switches in spare components or reassigns tasks among non-failed components]
Reinitialization
[checks, updates and records the new configuration and updates system tables and records]

Figure 3.4: Fault tolerance taxonomy from Avizienis et al. [27, fig. 16].

Given the research gap in the SLR and the researcher's background within software architecture, it was a logical step to limit the thesis scope to detecting *evasive* adversarial attacks using fault-tolerant architectures. The requirement for utilizing autonomous vehicles came from the prevalence of the scenario in common discourse and access to realistic datasets being readily available. Additionally, these datasets can be expected to be quite complex, accurately testing the architecture's ability to handle non-optimal scenarios. To limit the scope of the dataset, it was decided to use single-modal image datasets for this thesis. Reading the fault tolerance taxonomy in Figure 3.4 placed the implementation of these fault-tolerant architectures under the subset of *Error detection*. Referring back to the taxonomy in section 2.3, the architectures are intended to detect the presence of an activated fault. This would mean detecting the presence of adversarial images.

An important thing to note in this thesis is the use of the terms *classifiers* and *detectors*. Within machine learning, a classifier refers to a model which attempts to classify something, e.g. *classify* a cat as a cat. From software architecture, one has the concept of *detectors*, referring to a component that attempts to *detect* the occurrence of a specific set of circumstances. In the case of this thesis, these circumstances refer to the system being attacked by adversarial examples. As this thesis performs detection of adversarial attacks, the components performing this detection will be called *detectors*, despite actually being classifiers that only output the probability for one class. This class being 'How probable is it that this image has been attacked?'. The terms *classifier* and *detector* are used in this way to differentiate between machine learning models which try to *classify* an object and *detect* adversarial examples. This is done to reduce confusion and create a clear separation between these two concepts in the following sections.

As a result, the following research questions were chosen:

- **RQ1**: How can fault-tolerant architecture principles be applied to detecting adversarial attacks?
  - **RQ1.1**: How can multi-model architectures be used to improve the detection of adversarial attacks?
  - **RQ1.2**: How can diverse data be used in fault-tolerant architectures to improve the detection of adversarial attacks?

Chapter 4

Method

This chapter will present an overview of the different artifacts used in this thesis and the need for these artifacts in the thesis. The following section will denote the different tools used in the implementation of these artifacts. The subsequent sections will then detail and describe the implementation of these artifacts and how the tools were utilized to create them.

4.1 Prerequisite artefacts

To produce the fault-tolerant architectures and the technical environment in which to test them, a number of artifacts and implementations needed to exist. This section will present those artifacts and implementations, what need they fulfill, and what requirements were posed to them. In short, the required artifacts and implementations were:

- A benign dataset.
- An adversarial dataset.
- An implementation of a multi-model architecture detector.
- An implementation of a data diverse architecture detector.

The need for a benign and adversarial dataset came from the fact that the models inside the detection architectures needed to be trained. These two datasets would also be used to evaluate these detectors' performance. As mentioned in section 3.2, the benign dataset would need to depict a realistic and complex traffic situation. In terms of the adversarial dataset, additional requirements needed to be fulfilled. The need for a realistic adversarial dataset meant it needed to contain multiple attacks, as a defender could not expect to know what attacks they would be presented with. Additionally, these attacks could not come from the same 'family' of attacks, which meant that the attacks would be too similar. This would risk the detectors being too adept at generalizing for that subset of attacks, which could result in skewed results.

To answer RQ1.1, a multi-model architecture was required. Selected for this task was an implementation of the recovery block. This was due to its sequential nature but also allowed for investigating how performing multi-operation detection on the same input space could affect the detection of adversarial attacks. As mentioned in section 2.3, a sequential redundancy detector, such as the recovery block, functions best when each set of modules and acceptance tests iteratively filters out disjunct sets of adversarial attacks. Keeping this in mind, strict attention would have to be paid to the diversity of the component detectors. This is because diverse components are likely to have increased coverage.

To answer RQ1.2, required an architecture that would allow for testing how data diversity could affect an architecture detector. The selected architecture would be N-Version Programming (NVP). This is due to the fact that the architecture allows for multiple modules to perform transformations on the same input in parallel. This would then produce multiple input spaces for a given detector. Additionally, it would also allow for multiple combinations of input spaces, which would allow insights into how the composition of input spaces affects detection performance.

4.2 Tools

Several tools were used to generate the datasets and train the models for the thesis. This section will present the different tools and their role in this thesis.

Python

Python is a dynamically-typed general-purpose programming language produced and maintained by the 'Python Software Foundation' [43]. It supports multiple programming paradigms and has many popular frameworks using it as its coding language, such as Django [44] or Flask [45]. Its official package manager, 'pip', provides easy access to many useful libraries for machine learning via the package repository 'Python Package Index' [46]. The language was selected for this thesis due to the researcher's familiarity with the language, libraries provided by 'pip', and the sizeable documentation available.

Pytorch

Pytorch is a popular python-based machine learning framework maintained by The Linux Foundation [47]. The framework supports many features, making it easy to utilize the hardware for training models effectively. As a result, its ecosystem and tools have many projects which utilize its functionality as a base for efficiently interfacing with hardware [48]. Due to its great tools and support in the machine learning field, it was used for this thesis.

Pytorch Lightning

As mentioned, Pytorch is often used in other projects for interfacing with hardware. One is these Pytorch Lightning, a high-level open-source framework for training machine learning models [49]. It was initially created as part of a Ph.D. [50], it is now owned by the original creator under the firm 'Grid AI'. The framework utilizes its own ecosystem and classes to modularize models, which can be combined in different ways. In addition, the framework also has many useful and easily accessible features to ensure that training is performed correctly. Due to these features and its good integration with Pytorch, it was used for training the models in the thesis.

Adversarial Robustness Toolbox (ART)

The Adversarial Robustness Toolbox (ART) is a Python library hosted and maintained by The Linux Foundation [51]. The library is used to teach an understanding of how adversarial attacks work and contains a multitude of implementations of both attacks and defenses. These implementations are not limited to only *evasive* adversarial attacks either, and provide a solid resource pool for understanding how to create and defend against these attacks. Due to its extensive collection of implementations and support for generating adversarial attacks, this thesis used this framework to generate its adversarial attacks.

IDUN

IDUN is a server cluster run by NTNU's High-Performance Computing group [52]. The server cluster is available to students and employees at NTNU, that require access to additional computational hardware to perform research. The cluster consists of a number of nodes[1], with different hardware specifications. Utilizing the Slurm Workload Manager[2], it is possible to reserve a number of these nodes to perform specific processes. These processes, called 'jobs', can be called upon to run specific Python files. Of noteworthy per-user restrictions relevant to this thesis is the limit of 1TB of storage space per user. The relevance of this restriction will be discussed in section 4.5. Due to its sheer size and computational power, the IDUN cluster was used to train this thesis' models and perform other computationally heavy tasks.

4.3 The nuScenes dataset

As mentioned in section 3.2, the fault-tolerant approach for testing the architectures sets a requirement for using a complex and representative dataset. Given that an autonomous driving dataset was found, it would also have to be large, as large datasets allow for greater generalization of features, provide a lower risk of overfitting, and may be able to adjust for dataset imbalances [53] [54].

[1]https://www.hpc.ntnu.no/idun/hardware/. Accessed 2023-05-22

[2]https://slurm.schedmd.com/overview.html. Accessed 2023-05-22

One of the surveyed datasets fitting these criteria was 'nuScenes', a dataset often used in computer vision benchmarking [55], which depicts complex traffic scenarios [56]. Motional, the owners of nuScenes, have also hosted multiple object detection challenge workshops, the last of which was at the IEEE conference '2021 International Conference on Robotics and Automation 2021' [57]. Inspired by the multi-modal dataset KITTI [42], the dataset consists of 1000 driving 'scenes', where each scene is a 20-second snippet of a car in a traffic situation [56]. The 'scenes' are taken from two different cities, Boston and Singapore. Additionally, the scenes are taken at all hours of the day, in different environments (e.g., industrial and urban), and in diverse weather conditions. The vehicle-mounted sensor suite has six cameras, three pointing forward and three pointing backward. The vehicle is also equipped with five radars, three pointing forward and two backward. The sensor suite also has LIDAR and an Inertial Measurement Unit (IMU) to capture gyroscopic data. An overview of the sensor suite can be seen in Figure 4.1.

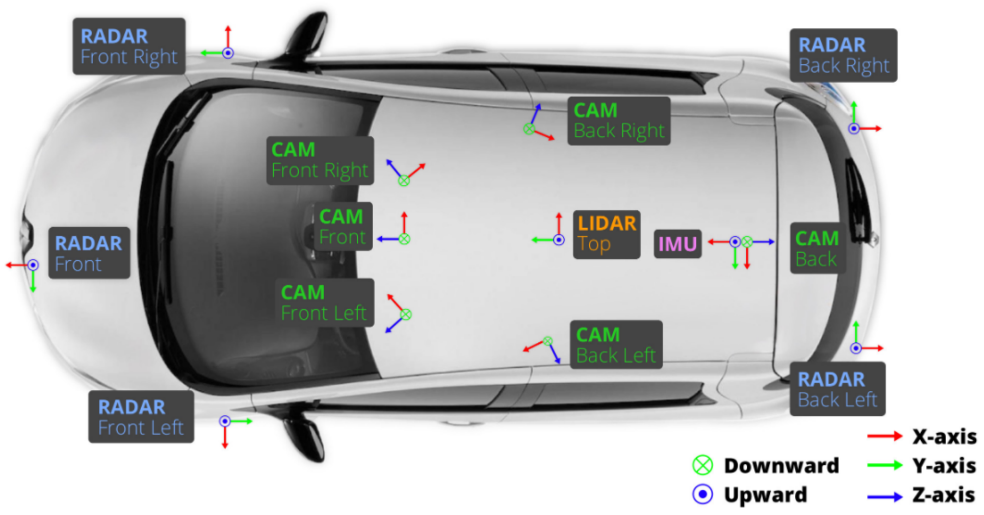


Figure 4.1: A figure showing the sensor array and placement on the cars from the nuScenes dataset. Adapted from [56, fig 4]

All of the cameras in the sensor suite capture and record images at a rate of 12 Hz. Of these twelve images, nuScenes denotes two of them as 'keyframes'. These 'keyframes' are annotated with all the objects in the scene. Additionally, these annotations contain useful metadata, such as how obscured objects are by other objects in the scene. For this thesis, only 'keyframes' were used as they contained annotations for objects. These annotated objects were also labeled as one of 23 classes. An example of such a class would be 'human.pedestrian.adult'. A number of these classes describe specialized instances of road users. E.g., multiple classes exist for describing pedestrians, such as 'human.pedestrian.construction_worker' or 'human.pedestrian.wheelchair'. An important thing to note is that an object can only belong to one class. This means that a police officer is only labelled as 'human.pedestrian.police_officer' and not additionally as 'human.pedestrian.adult'. The nuScenes dataset and tools contain support for the utilization of additional sources of information, such as LIDAR maps recorded with the LIDAR sensor shown in Figure 4.1. These maps were not used, however, as the thesis was focused on using images rather than sensor fusion methods. Still, it would be an interesting future work to see how sensor-fusion in fault-tolerant architectures would perform, mimicking the work by Yang, Karren et al. [41] on the KITTI [42] dataset.

4.4 Customized dataset

The nuScenes dataset was made with the intention of being a benchmark for real-time object detectors [56]. The dataset needed to be modified to suit the thesis' task, as the task enveloped detection. This section will describe the steps to produce the customized dataset.

4.4.1 Methodology

The first stage in producing the customized dataset consisted of three modifications:

- As shown in Figure 4.2, each nuScenes image contained multiple objects. To perform classification, the images would have to be cropped so that only one image contained one object. In addition to compatibility, cropping has also been shown to improve classifiers' ability to explore deep object features in images [58] and simplifies the classification of partially obscured objects [59].
- Objects more than 40 % obscured were removed from the dataset. This was done, as severe occlusion of objects is known to degrade models' ability to classify objects [60] [61].
- Images below 112 pixels in either their width or height dimension were removed. The size restriction was set because the ResNet classifier scales images to 224x224, and scaling of small images may enhance noise when upscaled [62]. There is no strictly defined 'lower bound' for how much an image can be upscaled before classification becomes a problem, as it can depend upon the complexity of an image and the task performed. However, based on works by Y. Pei. et al. [63] which trained multiple ResNet-50 classifiers, low-resolution images need to be downscaled by *more* than a factor of four before accuracy seems to drop significantly. The initial lower bound of 112, a factor of two, was set as a conservative value to avoid problems with a lack of spatial details.

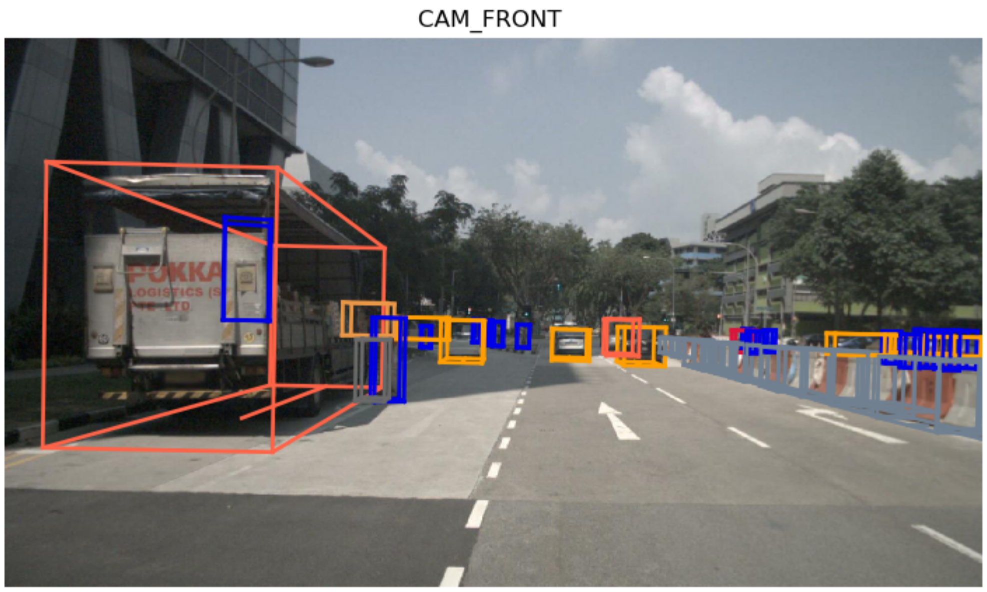


Figure 4.2: An example image from nuScenes showing multiple objects in a scene. Each bounding box represents an annotated object.

4.4.2 Implementation

In Figure 4.2, it can be seen that the provided camera annotations define the bounding boxes in 3d space. However, cropping these images meant the bounding boxes needed to be defined in 2D. To perform this conversion, a script[3] from the nuScenes GitHub page was used. Following this, a custom-written script used the 2D bounding box positions to crop the annotated objects from the larger images. This process is visualized in Figure 4.3. Using the visibility parameter provided by nuScenes, images more than 40% occluded were discarded. Pseudo-code for this process can be seen in algorithm 1.

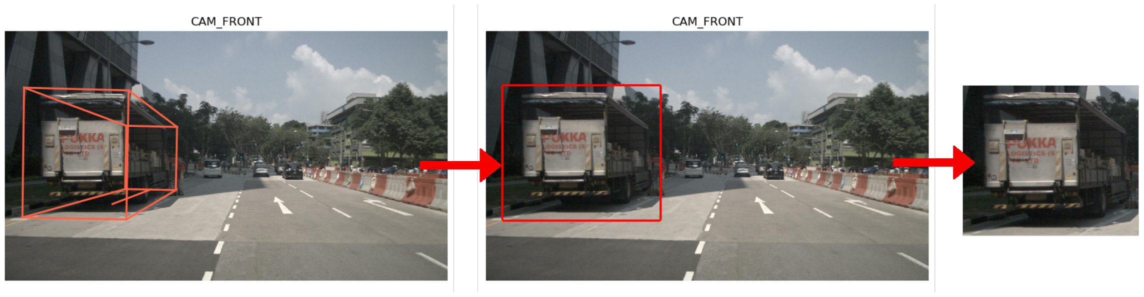


Figure 4.3: An image sequence showing the 3D to 2d pipeline. The initial 3D bounding box is converted to 2d, and the edges of the bounding boxes are then used to crop the image to only contain that object.

[3]Script available **here** | Accessed: 2023-05-22

The value proposed by the nuScenes dataset was its diverse and realistic scenarios. As a result, ensuring that the dataset retained its ability to accurately depict the different classes despite its variety, was essential. Otherwise, the dataset could be skewed in such a way as to produce invalid results. To verify that the modified dataset retained its ability to depict the classes, a ResNet-101 classifier [64] was trained and tested on the dataset to gauge its performance. ResNet models are trained to classify ImageNet objects and were thus likely to perform well on image classification in the nuScenes dataset as well. When enforcing the third requirement of the images needing more than 112 pixels in width and height, issues started to arise. After training the ResNet classifier, the classifier would perform significantly worse on some classes. In particular, the classifier performed significantly worse on classes that were in the minority of instances in the dataset. An overview of these results can be seen in Figure 4.4.

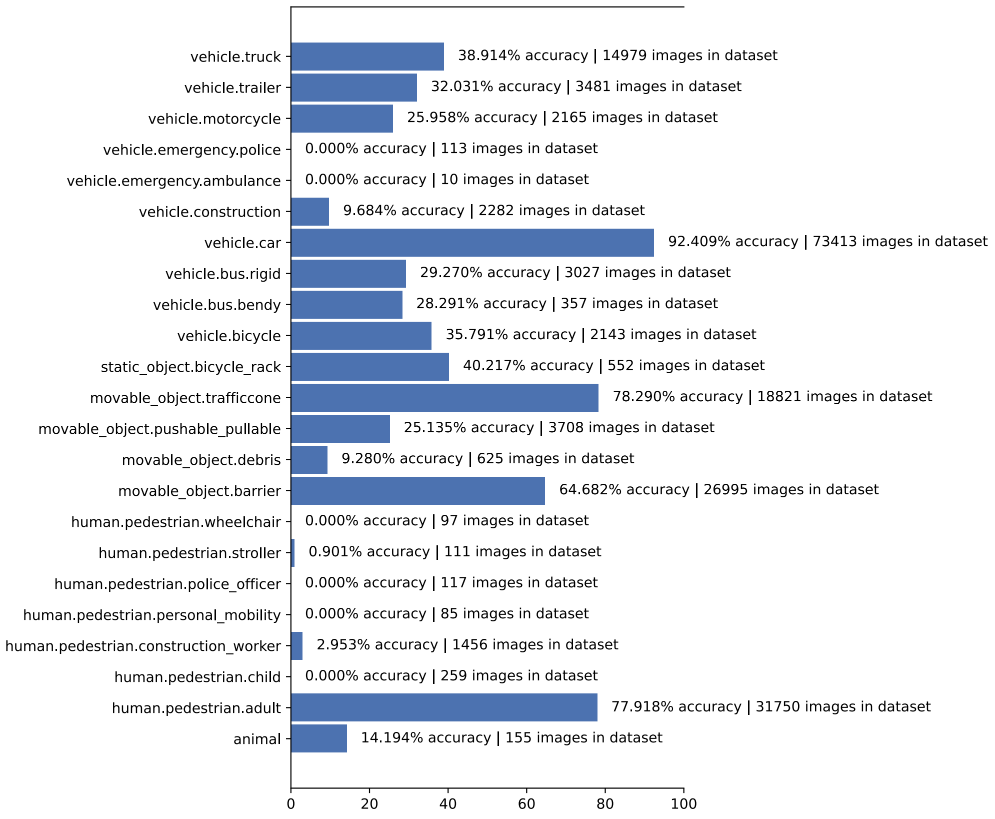


Figure 4.4: The per-class classification performance on the 23-class dataset.

When further investigations were conducted, it was observed that a large number of classes with a low number of instances were often classified within their own 'superclass'. E.g., instances of vehicle.emergency.police were often misclassified as 'vehicle.car'. This breakdown can be seen in Figure 4.5

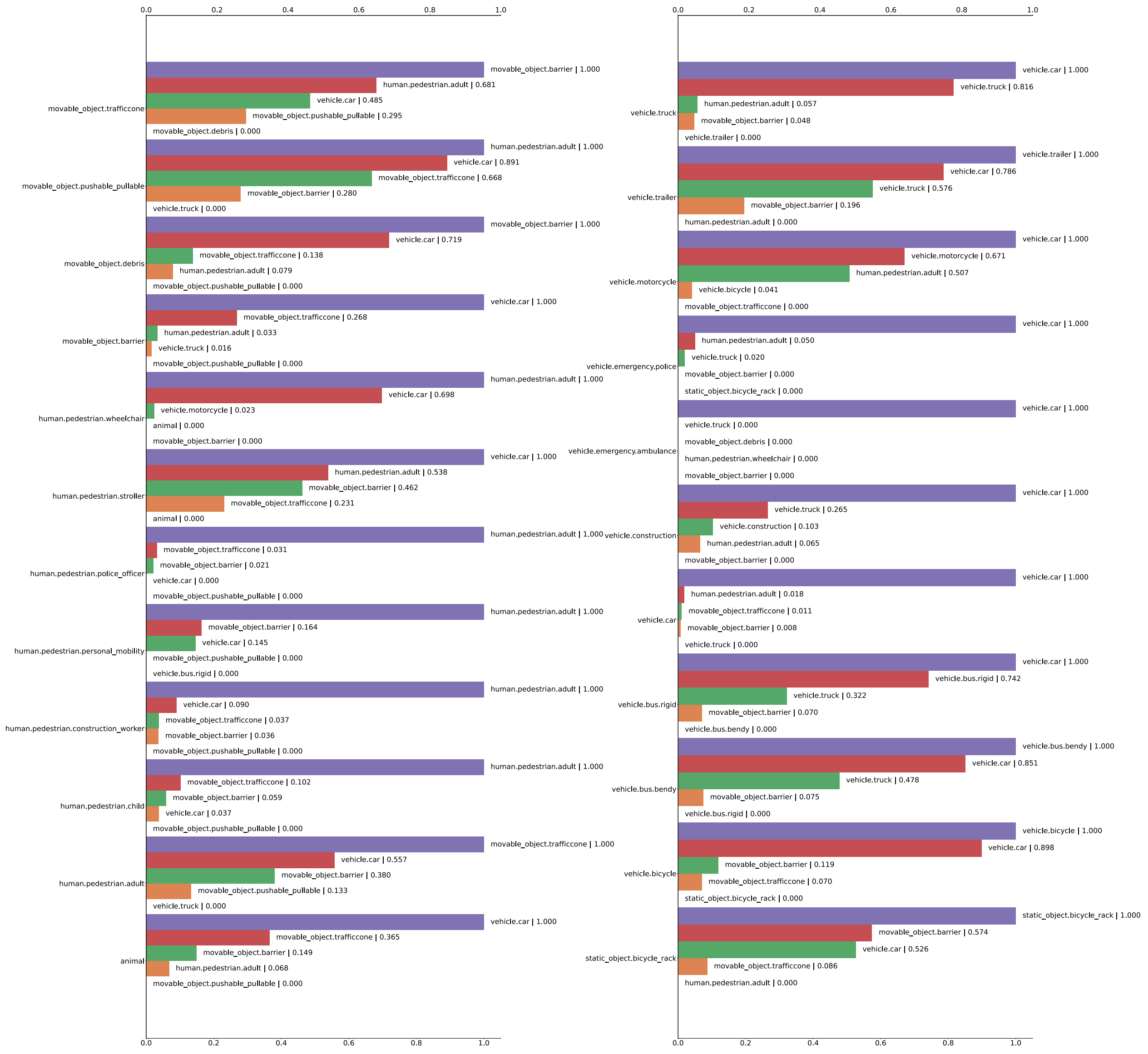
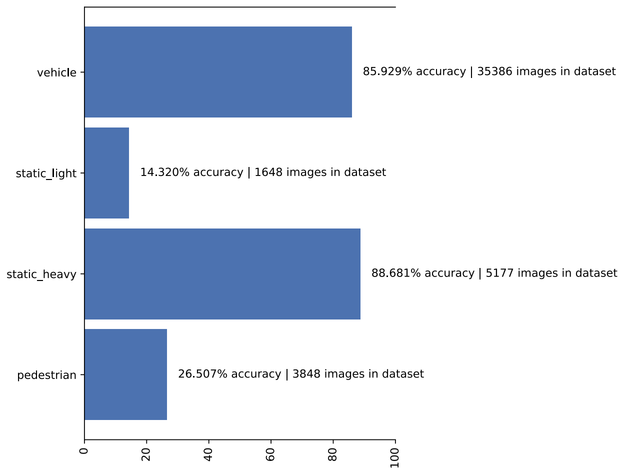


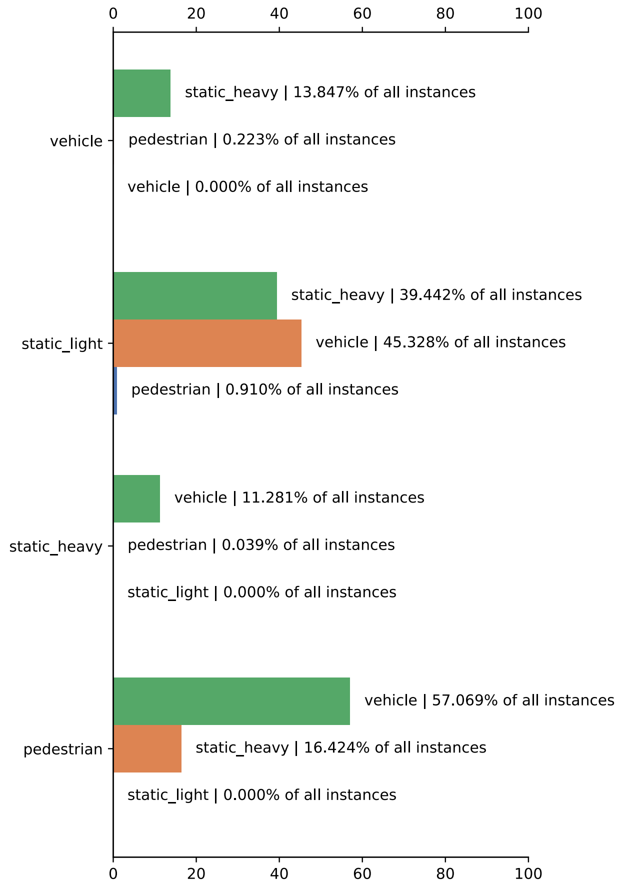
Figure 4.5: A plot showing the breakdown of the top 5 misclassifications, per class. Values are broken down by misclassifications in relation to the most misclassified class. This is to show how the top 5 misclassifications were distributed.

In order to mitigate this, a number of classes were merged, reducing the number of classes from twenty-three to four. The reasoning for why this could safely be done was that the badly-defined classes would *generally* be misclassified as their 'superclass'. E.g. human.pedestrian.wheelchair was often misclassified as human.pedestrian.adult. This can be seen in Figure 4.5. From the perspective of the vehicle, whether or not a pedestrian is in a wheelchair or not shouldn't significantly affect its behaviour. Thus, trading an increased number of classes for fewer, well-defined, classes seemed a reasonable trade-off. An additional note is that it was assumed that an adversarial attack on an autonomous vehicle wants to affect the behaviour of the vehicle. While misclassifying a walking pedestrian for a wheelchair user can *technically* be classified as a service failure, it would have little effect on a vehicle's behaviour. Merging the classes would make sure that a successful attack would affect the vehicle's actions and the dataset would contain well-defined classes. An overview of the mapping between the 23 classes and their new class in the 4-class dataset can be seen in Table A.1 in the appendix. The four classes are:

- Pedestrian. Includes all classes which are commonly found on the sidewalk, such as pedestrians, bicyclists and animals.
- Static_light. Includes static objects which pose little danger to the vehicle's passengers if hit, such as wheelie bins and traffic cones.
- Static_heavy. Includes static objects which can pose a danger to the vehicle's passengers it hit, such as construction barriers and garbage containers.
- Vehicle. Includes all forms of transportation vehicles which travel on the road, such as cars, trucks, ambulances and motorcycles.



(a) Per class accuracy.

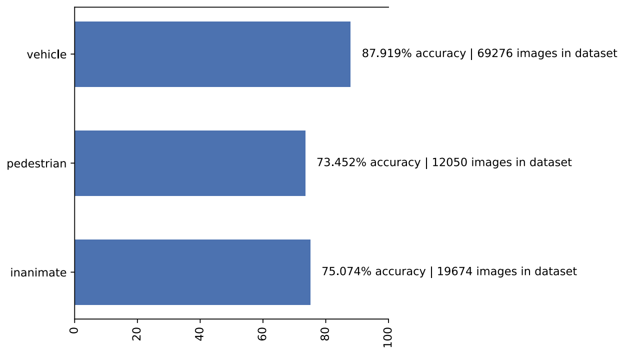


(b) Misclassifications per class.

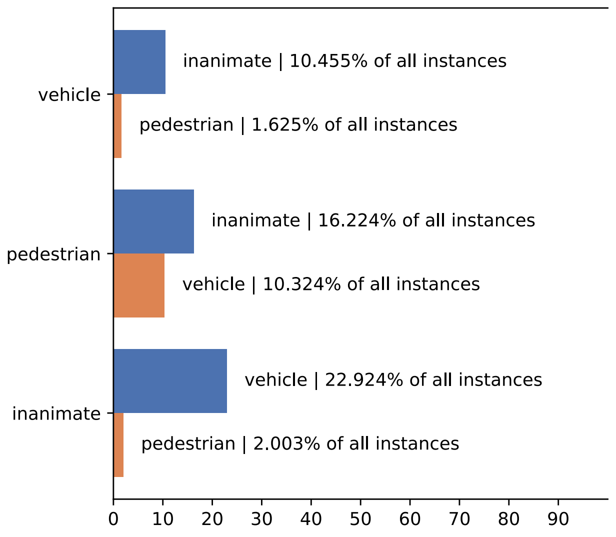
Figure 4.6: Performance and per-class misclassification plots for the transfer-learned ResNet-101 model. The number of images is based on the test dataset.

The ResNet classifier performed poorly during trials, as seen in Figure 4.6 a). The classifier performed strongly on vehicles and heavy static objects but poorly on light static objects and pedestrians. Both light static objects and pedestrians were consistently misclassified as either vehicles or heavy static objects as can be seen in Figure 4.6 b). This led to additional steps to ensure that each class would be well-defined. The first was the merging of the two static classes into a larger class called 'inanimate'. The mapping between the original 23 classes and the new three classes be seen in Table A.2 in the appendix. While this would reduce the information to a decision-making system, it was done to differentiate between objects which can be assumed to act statically rather than dynamically in a traffic scenario. Additionally, most static objects can be assumed to 'shape', change or otherwise impede traffic flow but not directly interact with traffic such as a pedestrian.

Additional steps were also taken to further improve classification performance. It was suspected that the imbalance between the number of images also affected the classification performance in favour of static heavy objects and vehicles. Given the requirement for images larger than 112 pixels, this imbalance was likely caused by these objects being closer to the nuScenes sensor vehicle. Being closer to the vehicle meant that the images were larger and thus were more present in the dataset. Additionally Keeping in mind that Y. Pei. et al. [63] showed that the accuracy of classifiers tended to drop off when trained on images scaled by more than a factor of four, it was considered safe to reduce this size restriction down to 64 pixels instead of 112 pixels. While it did perform better, the classifier still had some issues with per-class classification imbalance. This can be seen in Figure 4.7.



(a) Accuracy and samples per class



(b) Misclassification percentage per class

Figure 4.7: Performance of transfer-learned ResNet-101 model on the three-class data set.

Looking at Figure 4.7 (b), it is clear that the inanimate class and to some degree the pedestrian class were misclassified as vehicles. Understanding how this could occur is clearer when looking at pedestrian and inanimate images from the dataset. As exemplified in Figure 4.8, quite a few of the inanimate and pedestrian images had vehicles in the background or partially in the same region as the annotated object. This could also explain the misclassification of pedestrians as inanimate objects.

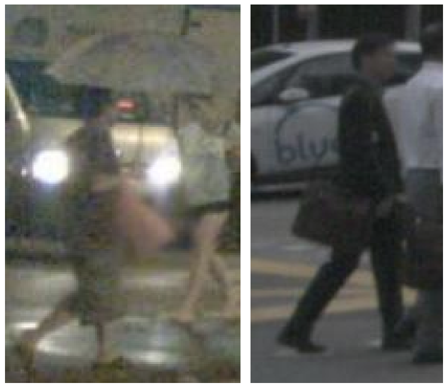

(a) (b)

Figure 4.8: Two examples of potentially difficult images classified as 'pedestrian'.

As shown in Figure 4.7, the 'vehicle' class had at least three times the number of images than the other two classes in the dataset. This was likely the cause of this performance imbalance [65]. Eliminating this imbalance involved truncating each class to 50 000 images each, split over their respective training, validation and testing datasets. The classifier's performance on the final benign dataset can be seen in Figure 4.9.

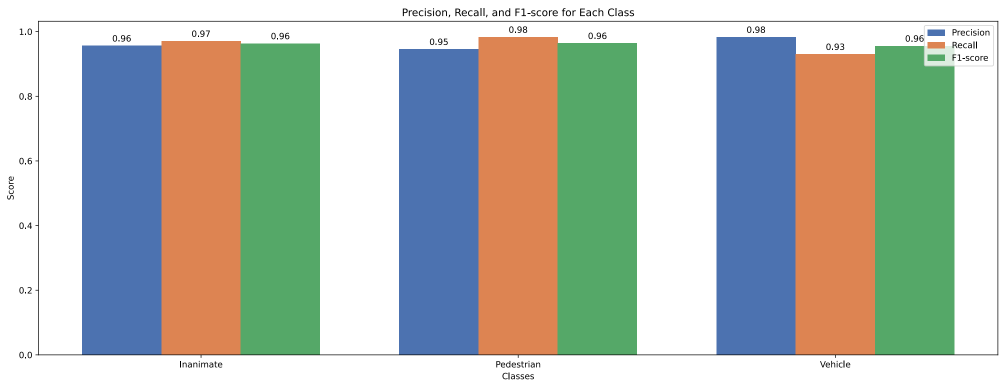


Figure 4.9: Performance of transfer-learned ResNet-101 model on three-class data set after reducing the number of vehicle photos.

4.5 Adversarial dataset

As mentioned in section 4.1, the adversarial dataset would be based on the customized benign dataset. Ensuring that the dataset would be realistic required multiple attacks to generate the dataset. Additionally, these attacks could not all come from the same 'family', as it could inadvertently skew the results. This section will present the selection of attacks and how these adversarial examples were generated.

4.5.1 Methodology

As mentioned in section 4.2, the Adversarial Robustness Toolbox (ART) was used to generate the adversarial images for the dataset. Four were chosen to generate the adversarial dataset from the many different adversarial attacks available in the framework.

- Fast Gradient Sign Method (FGSM) from Goodfellow et al. [17]
- Carlini & Wagner's L2 norm from Carlini et al. [66]
- Auto Projected Gradient Descent from Croce et al. [67]
- Shadow attack from Ghiasi et al. [68]

FGSM was chosen due to its prevalence in research papers. It is also quick to generate adversarial images, which made it useful for initial testing. FGSM works by calculating the gradient of the loss function in the classification model for a given input x. This is done by calculating the sign value, scaling it by a factor of ϵ, and subtracting this scaled sign value from the gradient. This produces a new adversarial image. The epsilon value is set in the implementation and dictates how much the attack is allowed to perturb the image.

The implementation of FGSM does support a targeted attack, such that it attempts to cause the reclassification of the image to another specific class. However, the attack was set to perform the untargeted version of the implementation, as the goal was to cause the misclassification of objects. What these objects were misclassified as did not matter.

Carlini & Wagner's L2 norm was also chosen due to its prevalence in adversarial attack literature as well as being noted as 'some of the strongest white-box attacks' by the ART-toolbox [69]. Another contributing factor to selecting Carlini & Wagner's L2 norm is due to their assertion that their attack is more potent on complex datasets [26], such as the one generated for this thesis. Additionally, Carlini & Wagner's L2 algorithm is not based on FGSM, which was reasoned to create perturbations dissimilar to FGSM. Thus, selecting Carlini & Wagner L2 would also support the effort of producing a realistic adversarial dataset.

The attack attempts to find the smallest perturbation that will change the classification of an image, given the model's gradients. The different implementations of Carlini&Wagner attacks use different norms and methods to calculate the perturbation. The constraints dictated by the parameters again bound these perturbations. Similarly to the FGSM attack, the Carlini-Wagner L2 attack was run in the untargeted mode.

Unlike the first two attacks, published in 2014 and 2016, the final two attacks were chosen due to them being the two most recent additions. The reasoning was that due to acceleration within the field of adversarial attacks, FGSM and Carlini L2 norm would not necessarily be representative of the current state of adversarial attacks. Thus, these two were added to ensure that one would not create a disadvantage in favor of the defenders by utilizing old

FSGM Parameter name	**ART framework default**	**Dataset value**
Epsilon	0.3	0.05
Epsilon step size	0.1	0.0001
Distance Perturbation Norm	L-infinity	L2
Auto-PGD parameter value	**ART framework default**	**Dataset value**
Epsilon	0.3	0.05
Epsilon step size	0.1	0.0001
Distance Perturbation Norm	L-infinity	L2
Carlini&Wagner L2	**ART framework default**	**Dataset value**
Max Iterations	10	20
Max number search halvings	5	10
Max number of search doublings	5	10
Shadow attack	**ART framework default**	**Dataset value**
Number of steps	300	600
Attack learning rate	0.1	0.01

Table 4.1: Table showing the ART framework default parameters and the updated values for the dataset.

attack techniques against newer detection techniques. This would also diversify the attack dataset, as the introduction of newer attacks could produce distinct adversarial perturbations compared to the older attacks.

Auto Projected Gradient Descent (APGD) is within the ‘family’ of FGSM-attacks and is functionally the same as Projected Gradient Descent [67]. However, it goes on to add some algorithmic optimizations. PGD works by initializing its perturbations uniformly, iteratively applying FGSM, and clipping values exceeding the maximum allowed perturbation size ϵ. After calculating the loss, the attack performs a step in the opposite direction of the model gradients. In PGD, this step size is static. What APGD does differently is applying a non-static step size to the attack in addition to utilizing a different loss function to calculate the loss. As noted in their paper, these optimizations address issues where the algorithm’s high dependency on an appropriately small step size could cause the model to struggle to find a global maximum.

Shadow attack is an optimization-based attack that functions similarly to PGD or FSGM but differs by having several constraints which aim to reduce how perceptible the perturbations are in the image. Unlike APGD, where the perturbations are clipped using some maximum perturbation scale, shadow attack applies three penalties to three characteristics of the perturbations. These three penalties penalize perturbations with significant pixel variations in a small space, induce a large global change in the per-channel mean, and introduce disproportionate changes to individual color channels on a per-pixel basis. The paper claims this will result in small, smooth perturbations with non-drastic color changes [68].

A thing to note is that only white-box attacks were utilized to generate the adversarial attack dataset. This is due to the fact that black-box attacks need to estimate the gradients of the detection model [19]. This is generally done by attacking the model and incrementally improving the estimated gradients based on the output from the model [70]. In the early stages of the black-box attacks, this could have resulted in weakly attacked images in the dataset. Sticking to white-box attacks would ensure that the images in the adversarial dataset were proficiently attacked.

The parameters used to generate the attacks were largely set to the default parameters from the Adversarial Robustness Toolbox. However, some of the default parameters in the toolbox were considered to produce far too overt attacks, such as the default epsilon value of 0.3 for FGSM. This had the potential to cause perturbations too obvious and thus be too easy to learn for a detector. As a result, several parameters were adjusted to create more complex and subversive images. These changes from the default values are listed in Table 4.1.

An important point to note is that adversarial images which did not successfully trick the classifier were not filtered out. This decision was based on the fact that, in the scenario, the detector would be assumed to be a part of a larger software architecture. In terms of fault tolerance, removing the unsuccessful attacks would improve the error detection if a dormant fault was activated. This is because the detector would be trained to detect successful attacks. However, viewing the detector as part of a larger system would have made it more useful to alert the system when someone is attempting to attack the system instead. Knowing when the system is under attack would allow it to take preventative measures before the attacks become successful. One such preventative measure could be stopping the vehicle. Assumed part of a larger architecture, the unsuccessful attacks would be retained in the adversarial dataset.

4.5.2 Implementation

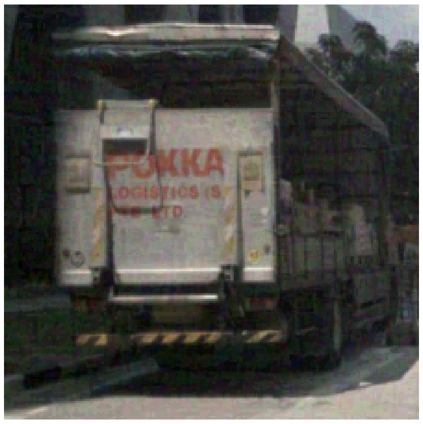
During the generation of the adversarial dataset, Carlini&WagnerL2 and ShadowAttack proved too resource intensive to effectively produce 150 000 adversarial examples. As a result, the number of adversarial images was reduced to 15 000 per attack. This would result in an adversarial dataset with 60 000 images.

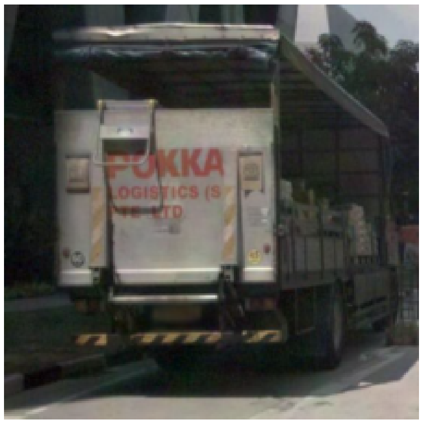
As noted in section 4.2, the adversarial images were generated using the IDUN server cluster. This was done because the local hardware available was insufficient to produce the images in a reasonable timeframe. Even when having access to IDUN, the images had to be processed in parallel, where one job would produce images for a specific class in a specific dataset. That is, one job would produce only Carlini&Wagner L2 images for the 'pedestrian' class in the 'train' dataset. Given unlimited access to computational resources on IDUN would have made this process take about 24 hours to produce the dataset. However, due to queues, per-user restrictions, and I/O bottlenecks, generating the 60 000 images took about four days.

An important note is that the images were not stored as either numpy arrays in the '.npy' format, compressed numpy arrays in the '.npz' format, or any other array format such as pickle. This was due to the previously mentioned file restrictions noted in section 4.2. Storing the adversarial dataset in any of these formats would immediately result in saturation of the 1TB per-user quota. Instead, the adversarial images used for training and testing were stored as '.tiff' images using OpenCV's Python implementation. Choosing '.tiff' as the intermediary format would allow the adversarial images to be stored as 32-bit float values, the same data type as the PyTorch tensors defining said adversarial images. Further, the libtiff library, which OpenCV uses for '.tiff' files, allows the images to be saved with a lossless algorithm being applied, thus ensuring that little to no data would be lost due to the intermediary storage stage. While these images would by no means be small, each being $\approx$ 600 KB, it was small enough not to saturate IDUN's file storage restrictions. Examples of the produced adversarial examples can be seen in Figure 4.10. Additionally, pseudo-code for the production of these images can be seen in algorithm 2 in the appendix.

(a) APGD

(b) Carlini&Wagner L2


(c) FGSM


(d) Shadow attack

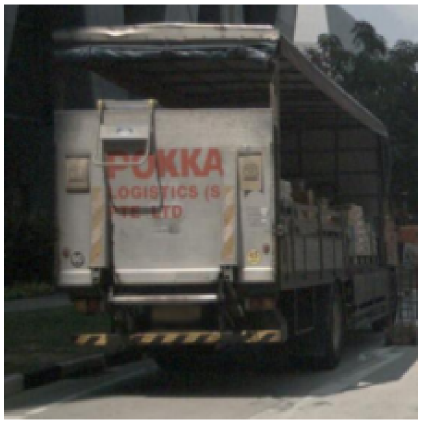

(e) Benign

Figure 4.10: Examples of generated adversarial images next to the benign version of the image.

4.6 Architecture 1: Recovery block

As mentioned in section 4.1, the thesis required the implementation of a recovery block detector. This section will present the implementation of this architecture in addition to presenting the steps carried out.

4.6.1 Methodology

As mentioned in section 2.3, both Avizienis et al. [27], and McAllister et al. [28] note the importance of design diversity and disjunct detection as keys to the success of sequential redundancy architectures. As a result, attention was paid to ensuring that the modules would be structurally diverse.

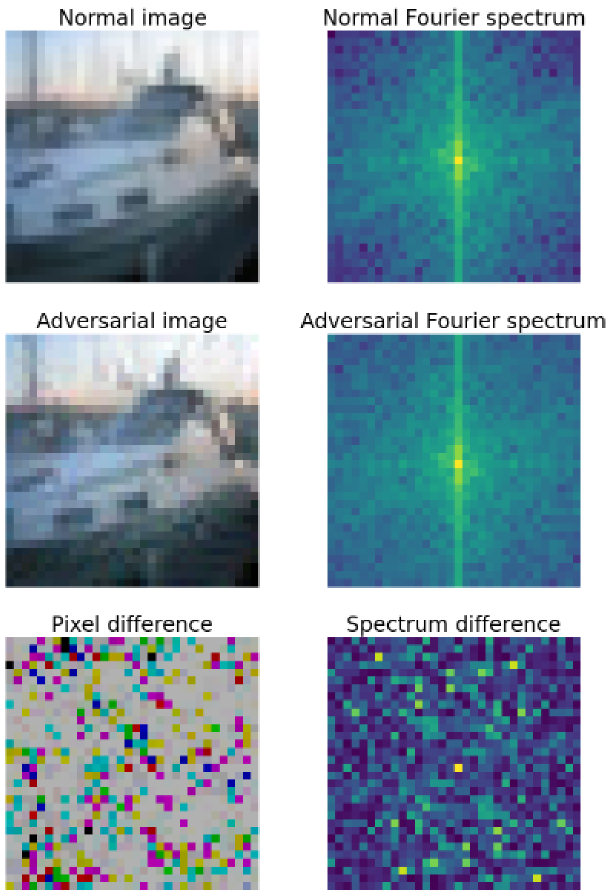


Figure 4.11: A visual example of the spectral detector.
Adapted from: Paula Harder et al. [38] figure 1.

The goal of the recovery block was to understand how multi-model configurations can affect the coverage of a fault-tolerant architecture. As mentioned in section 2.3, the two factors affecting this are the effectiveness of the operations and the applicable domain in the input space. The redundancy block aims to observe how these improvements can be implemented and how the improvements to operations affect the domains in the input space.

The first module in the recovery block was a detector from Paula Harder et al. [38]. As mentioned, the detectors transformed the images using a Discrete Fourier Transform (DFT). The transformed images would then be passed to a logistical regression model, which would make a prediction based on some metric. A visual representation can be seen in Figure 4.11. Paula Harder et al. [38] tried multiple implementations of this methodology, achieving strong performance on some attacks. One of these models, 'InputMFS', performs its detection based on the input image's magnitude within the Fourier spectrum. It is not the best-performing detector in the paper, but it did have some interesting properties. Despite its simple architecture, it showed varied performance across different attacks. I.e., It showed strong performance on attacks such as FGSM or BIM but not as strong performance on Carlini&Wagner-attacks. As recalled from section 3.2, the goal was to produce representative models which allowed for comparing the architecture and the per-component performance. This diversity in per-attack performance was desirable, as varied performance across attacks could provide insights into how fault-tolerance coverage propagates in sequential redundancy architectures.

The spectral detector was quite shallow, only relying on a single logstistical regression layer for classification. To contrast this, the second detector would have to be large, i.e. consist of a large number of layers and parameters. This was due to the fact that the key to increasing coverage in fault-tolerant architectures is design diversity. Using large models against adversarial attacks is also noted to have some inherent advantages. As indicated in A. Kurakin et al. [71], models with an increased number of parameters tend to be more robust against adversarial examples. This is also supported by the works by A. Madry et al. [23], where they note that training a ResNet-model on only natural examples its robustness scales with an increase in parameters. This effect is especially notable on adversarial examples with smaller perturbations [23]. While both papers discuss adversarial *training* and not *detection*, it points to models with more parameters being more capable of generalizing adversarial noise. This would also mean that the model should be more capable of differentiating between benign and adversarial images. As a result, the second detector was decided to consist of a high-capacity, pre-trained, and transfer-learned ResNet-101 model. From here on, this detector is referred to as 'the binary detector'.

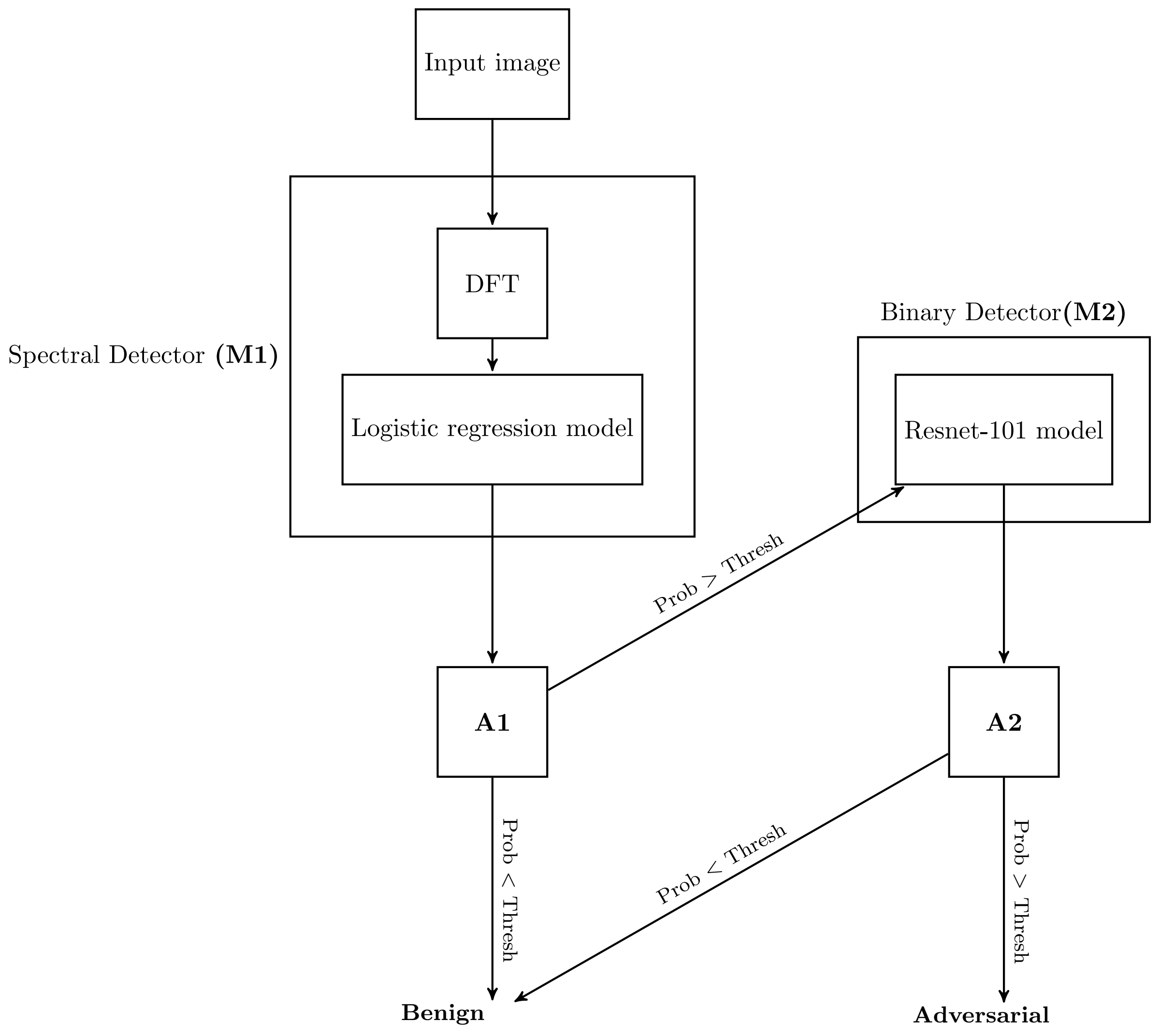


Figure 4.12: A figure showing the Recovery block architecture. In both acceptance tests A1 and A2, the confidence value is checked and passed on based on if the probability of it being an adversarial image is above or below the threshold. The model is meant to replicate the architecture of the recovery block in Figure 2.5

A figure depicting the implemented recovery block detector is shown in Figure 4.12. It depicts the spectral detector was set as the first module (M1), with the acceptance test (A1) being a threshold that determines if the image is benign or adversarial. If an image crosses the threshold set in A1, the image would be passed to the binary detector (M2). This also produces a probability of the image being adversarial. Based on this, the second acceptance test (A2) would see if this probability exceeded its threshold and classify it appropriately. In a real-world scenario, these thresholds would have to be determined beforehand. However, in this thesis, the thresholds would be calculated to produce the best F1 score based on the predictions for the test dataset.

The rationale for the presented model configuration was based on the fact that the spectral detector was assumed to perform differently on different attacks and thus pass a varied distribution of attacks to the binary detector. In essence, it was done to induce disjunct sequential detection between the models. In accordance with the implementation shown in M.R. Lyu et al. [28], the architecture was implemented in a *blacklist*-configuration, whereupon the default state is to classify an image as benign, requiring both models to classify it as adversarial for the detector as a whole to detect it as adversarial. An argument could definitely be made for its implementation in a *whitelist*-configuration, where each image needs to be classified as benign twice to be used as a benign image. In a realistic scenario, the selected configuration would have to be chosen based on the components' capabilities and the system's non-functional requirements. However, neither of these were known at the implementation stage of these architectures. As a result, the recovery block was implemented in a *blacklist*-configuration, in adherence to the implementation from the 'Handbook of Software Reliability Engineering'.

4.6.2 Implementation

As mentioned in section 4.2, the models were trained using the PyTorch Lightning framework. While this simplified the training process and ensured reproducible results, it also required the re-implementation of some of these models. While the authors of the 'spectral detector' did provide a GitHub repository of their implementation, it was written in the Tensorflow machine learning framework instead. While it was possible to utilize their Tensorflow implementation, implementing the detector in the Pytorch framework would have ensured consistent usage of frameworks and systems for evaluating the fault-tolerant architectures. The relative simplicity of the detector also made it relatively simple to re-implement the detector in the Pytorch framework. Using pytorch's transforms, the images were applied the DFT, and their Fourier spectrum was retrieved. The spectrum was normalized per batch and flattened to a 1D tensor as it needed to be passed to a logistical regression model. The transformed images were used to train the detector using this Pytorch implementation of the 'spectral detector'.

The implementation of the binary detector went quite similarly, with the implementation of a Pytorch lightning model loading in the pre-trained ResNet-101 weights. The 'binary detector' fully connected layer would be changed to output one class. This class would indicate the likelihood that an image would be adversarial, going from benign with 0 to adversarial with 1. An additional thing to note is that all images were interpolated using Pytorch's 'interpolate' function to ensure that the images had the required ResNet pixel dimensions of 224x224. Careful note was paid to use anti-aliasing and non-alignment of corners, as per the documentation it would be equivalent to the standard Pillow sampling operation for resizing images.

	Early stopping epoch	Learning rate	Optimizer
Spectral Detector	39	0.001	Adam
Binary Detector	5	0.0001	Adam

Table 4.2: Table showing the epoch in which the early stopping callback was triggered, learning rate, and optimizer used for the Binary detector and the spectral detector. Models were trained with a batch size of 32.

The models were trained in a heuristic fashion, in which the models were tested with the learning rate at each magnitude and its halfway point to the next magnitude for values between 1x10^{-3} and 1x10^{-6}. E.g. 1x10^{-3}, 5x10^{-4}, 1x10^{-4} etc. Further, a regularization method called 'early stopping' was used to determine the appropriate time to stop the model training [72]. This method would check the model's performance on a validation set every epoch and stop the training if the model did not show improvement over two epochs. Training the model in such a way was done to avoid outfitting on the training set and producing models representative of a detector [72]. The hyper-parameters used can be seen in Table 4.2.

During training and testing of the recovery block detector, careful note was paid to the repeatability of the results. Both processes were seeded using Pytorch Lightning's seed function, which, given a seed, causes the training process to be pseudo-random but deterministic. As the benign and adversarial datasets numbered 150 000 and 60 000 images respectively, the benign dataset would have to be truncated down to 60 000 images to ensure an even balance of benign and adversarial images. Seeding the training and testing processes ensured that these would be deterministic.

During the testing of the recovery block detector, the weights of the two trained models were loaded in, and the benign and adversarial datasets were mixed. This would be followed by the images being sent to both detectors and both detectors producing prediction probabilities. In a realistic scenario, only the images detected as adversarial by the spectral detector would be passed to the binary detector. However, as mentioned previously, the optimal threshold for knowing when the spectral detector should pass the images to the binary detector was not known at run-time. Thus, the thresholds were calculated by generating 500 thresholds and calculating which threshold provided the best F1 score for the detectors. Utilizing python for calculating the optimal threshold for an individual detector was trivial but proved too computationally expensive for calculating the two thresholds in the recovery block. This was due to the fact that calculating optimal thresholds for two models is essentially a grid search, exponentially increasing the run time. Thus, a rust script was utilized instead of python to calculate the two optimal thresholds for the two detectors inside the recovery block model0.

4.7 Architecture 2: N-Version Programming

As noted in section 4.1, the N-Version Programming (NVP) architecture was implemented to assist in answering the question posed in RQ1.2. This section will describe the methodology and reasoning behind the NVP architecture and present how the process of implementing the architecture was performed.

4.7.1 Methodology

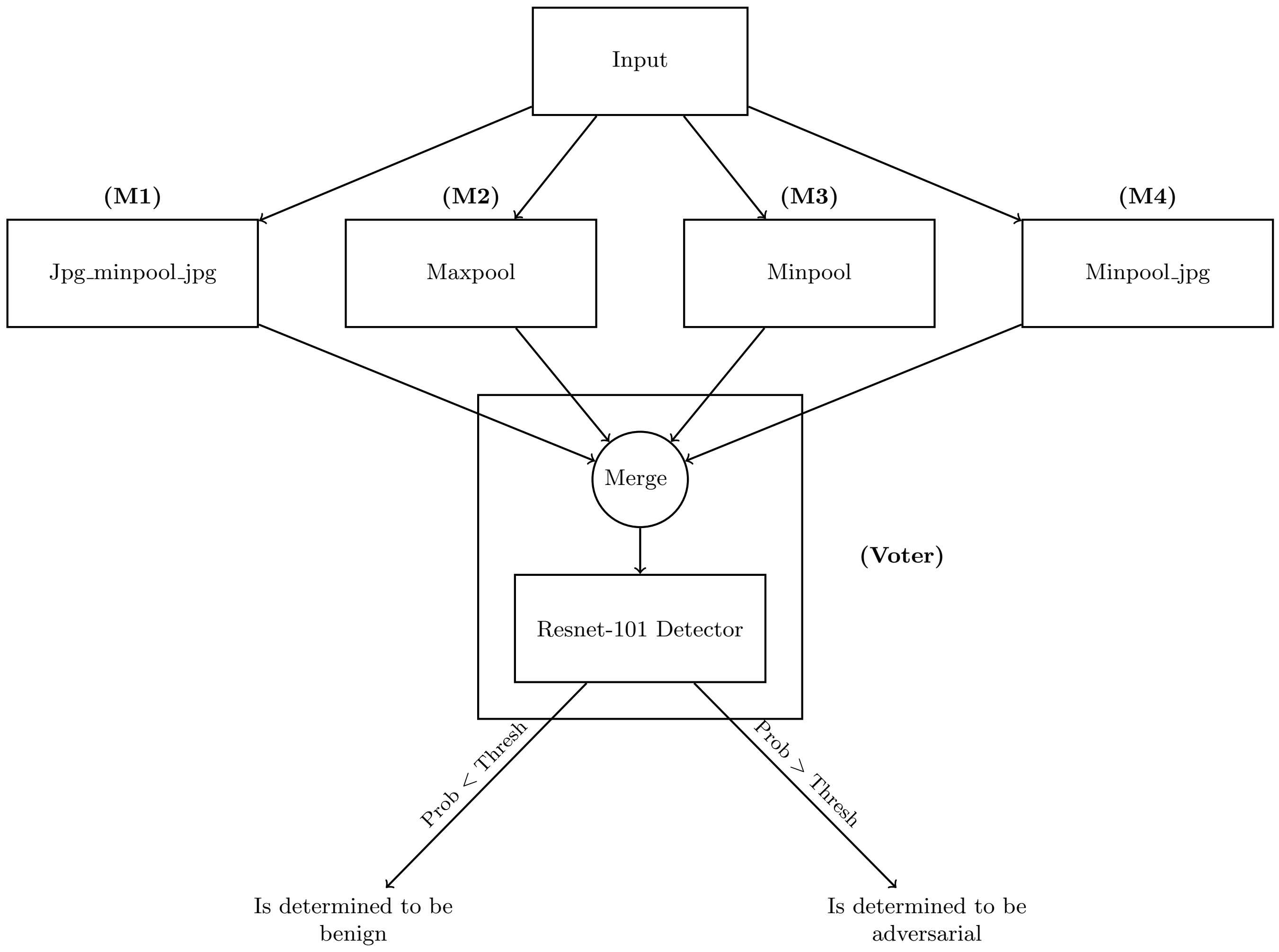


Figure 4.13: A figure showing the architecture of the N-Version Programming detector. The 'merge'- node represents that the four transformed image tensors with shape 3x224x224 are merged into a 12x224x224 tensor before being sent to the detector. This model is meant to replicate the NVP architecture in Figure 2.6.

As mentioned in section 3.2, the N-Version Programming architecture was intended to explore how data diversity affect can improve detection of adversarial attacks. As mentioned previously, using a shift in the *input space* could be used to increase the *domains* in which to detect attacks. In a traditional n-redundancy architecture, each set of modules would have its own input space and perform operations with the same objective. However, NVP being a software implementation n-redundancy means that it is not possible to change the input space of the architecture as a whole. However, it is still possible to test how input space affects coverage by looking at the performance of the voter instead. Thus, by treating each of the modules as a separate input space for the voter, it is possible to observe how a shift in input space can affect the fault tolerance of a system.

An overview of this thesis implementation can be seen in Figure 4.13. Each module would aim to transform the image in a way that was known to be advantageous for detecting adversarial attacks. The different module transformations were based on the results from a paper in the preparatory project Mekala, Ronan et al. [73]. In their paper, they perform a number of different transformations on images and compare the metamorphic distances between the original and transformed images. Metamorphic distance referring to the difference in the semantic data in the image, not easily changed by transformations such as e.g. rotation or scaling. The paper identifies a number of non-linear transformations which create a detectable shift in the L2 distance of an image. While the implemented N-Version Programming detector does not compare the metamorphic distances between the original and the transformed image, it was reasoned that if these transformations were able to create a detectable metamorphic shift, then a deep neural network would also be able to generalize for these metamorphic shifts. As a result, the N-Version Programming detector would take the four best-performing transformations. Adjusted for this thesis, these transformations were:

- Minpool: Applying a minpool kernel to the image.
- Maxpool: Applying a maxpool kernel to the image.
- Minpool-JPG: Minpooling and performing JPG-compression with 10% compression strength.
- JPG-Minpool-JPG: Performing JPG-compression with 10% compression strength, applying a minpooling kernel and then again performing JPG-compression with 10% compression strength.

Each module would transform the input image at run-time using its transformation operation. The voter would take the four transformed images, being 3x224x224 tensors, and merge them along their channel axis, resulting in a 12x224x224 tensor. This tensor would then be passed to a transfer-learned ResNet-101 model. Similarly to the binary detector, it was assumed that such a large model would have a larger capacity for generalizing adversarial noise.

In addition to the NVP detector, four additional detectors would also have to be trained. Unlike the NVP detector, each of these four detectors would be trained on only one of these four transformations. This was done to isolate the performance of any given transformation. Without these detectors, it would be difficult to discern whether or not any given performance increase was due to the architecture or the given transformation being optimal for detecting adversarial attacks. Additionally, it would also give some relative context in terms of the performance of the NVP-detector, as understanding how well the underlying detector performs with transformed images gives a benchmark in which to compare performance improvement provided by the NVP-detector.

In furtherance of the goal of exploring how data variety can enhance the performance of architecture models, detectors with access to only two of the four transformed images will also be trained. Henceforth known as ‘dual-transform’ detectors, these would instead be trained on 6x224x224 tensors to see how the different combinations of transformations affect the performance of the detection model. Tied into the results from the individual detectors as well as the NVP detector, it would also provide the ability to see how performance changes as the input space to the voter is increased.

4.7.2 Implementation

The training of these models was performed similarly to how it was done with the recovery block. Utilizing the PyTorch lightning framework, a custom implementation was written for each of these models and respective transformations, which would then be passed to IDUN to perform the actual training. The benign images were truncated down to the size of the adversarial images and mixed together, with benign images labeled as 0 and adversarial images labeled as 1. Following this, four transformed images per benign image were generated by performing the four module transformations. The images were then concatenated along the channel axis such that four 3x224x224 images became one large 12x224x224 tensor. This tensor was then passed to the detector inside the voter for training. An important thing to note is that, unlike the recovery block detector, the number of epochs of no improvement before the training was stopped was set to five. This is because the voters were transfer-learned and utilizing transformations that would significantly change the image. In order to induce the detectors to search for optimal points outside of the local area of the original weights, this value was increased to induce this behavior.

A similar process would be performed for the individual and the dual-transform detectors. Of note is that these models would have a similar but slightly different class implementation, as they would have to be trained on one and two transformations, respectively. The utilized parameters can be seen in Table 4.3. Similarly to the recovery block, the hyper-parameters were tested heuristically, with the best-performing parameters being used. The disparity between the optimal learning rates for the individual detectors and the NVP detector is assumed to be due to their disparity in input spaces. Going from a 3x224x224 tensor to a 12x224x224 tensor, and four different transformations to observe patterns, it was not seen as unreasonable that the optimal learning rate would differ to some degree. Another thing

	Early stopping epoch	Learning rate	Optimizer
NVP detector	18	0.00005	Adam
Jpg_minpool_jpg	6	0.0001	Adam
Maxpool	8	0.0001	Adam
Minpool	8	0.0001	Adam
minpool_jpg	11	0.0001	Adam

Table 4.3: Table showing the epoch in which the early stopping epoch callback was triggered, learning rate, and optimizer used for the NVP detector and its individual detectors. Models were trained with a batch size of 32.

	jpg_minpool_jpg	**maxpool**	**minpool**	**minpool_jpg**
jpg_minpool_jpg	▨	6	8	17
maxpool	13	▨	11	26
minpool	22	26	▨	17
minpool_jpg	13	8	12	▨

Table 4.4: Matrix showing the early stopping epochs for the dual-transform detectors. Rows indicate the transformation occupying the first three channels, and the columns indicate the transformation occupying the last three channels in the 6x224x224 tensor. Models were trained with a batch size of 32.

to note for the dual-transform detectors is the additional value provided by the increase in epochs before the early stopping callback was triggered. The models utilized pre-trained weights, which would risk the models naturally favoring the first three channels rather than the newly added last three. With the increase up to five epochs, it was assumed that it would allow the detectors to start exploring the utilization of the newly added channels to increase their detection performance. The dual-transform detectors were trained with the same hyper-parameters as the individual detectors, and their early stopping epochs can be seen in Table 4.4. An additional thing to note is that the dual-transform detectors were not trained with the same transformed image for the first and last three channels, as this would only duplicate the input space and not increase it.

The NVP-detector, its four individual detectors, and the dual-transform detectors were tested similarly to the recovery block testing in section 4.6. The model weights were loaded, transformations performed, and predictions made. Additionally, the same process of saving all predictions and calculating the optimal threshold for the predictions was done in the same manner.

Chapter 5

Results

This section will present the results and performance of the recovery block and the N-Version Programming detector after having tested their performance on the test set. The recovery block section will compare the error detection of the recovery block to the spectral detector and binary detector as if the latter two detectors were tuned to act as individual detectors. This was done as the threshold of a detector inside and outside the recovery block could require very different tuning for optimal performance. The N-Version programming section will compare the results from the NVP detector to the individual detectors. Additionally, the dual-transform detectors will also be introduced and present additional insights.

The metrics used to evaluate these models were ROC-AUC and F1-score. These metrics were selected, as they are heavily used in evaluating *classifiers* and assess two different performance aspects. ROC-AUC maps the true positive and false positive rates for all thresholds and calculates the area under the mapped false positive and true positive rates [74]. This means that ROC-AUC weighs benign and adversarial images equally. However, in a realistic scenario, the ratio of benign and adversarial images cannot be assumed to be the same. Acting as a counterweight to ROC-AUC is the F1 score, which measures the harmonic mean between precision and recall based on only one threshold. In terms of realistic operation, the F1 score is more applicable, as any given detector only has a single threshold upon which to base its detection.

5.1 Recovery block detector

The performance of the recovery block, spectral, and binary detectors can be seen in Figure 5.1. The figure shows the performance of the recovery block detector, the spectral detector and the binary detector measured in both F1-score and ROC-AUC. As can be observed, the binary detector performs notably worse than the spectral and recovery block detectors in both F1 and ROC-AUC. Further, the recovery block and the spectral detector perform quite similarly, with the recovery block showing slight improvement over the spectral detector in the F1 score and the spectral detector performing slightly better in ROC-AUC. That the spectral detector performs better in ROC-AUC is not necessarily surprising, considering that the recovery block also utilizes the worse-performing binary detector in its error detection architecture.

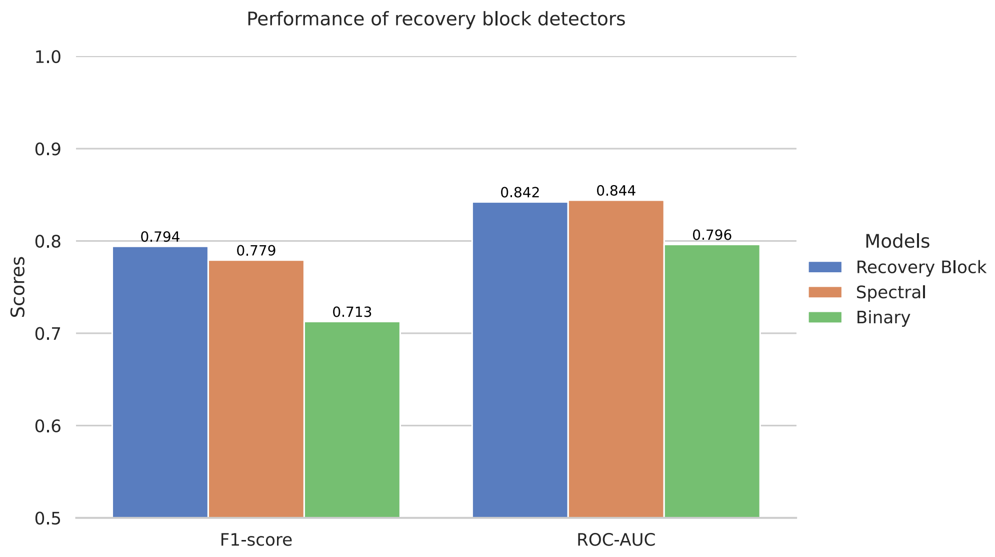
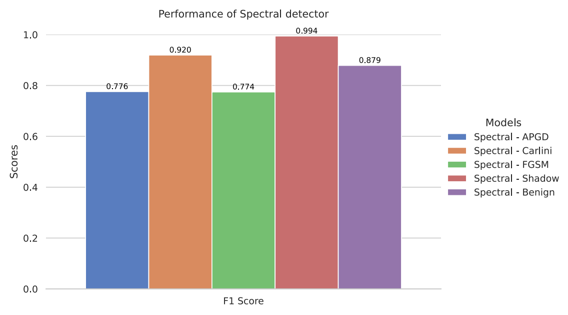


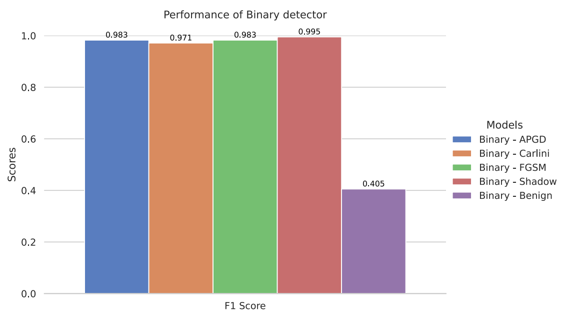
Figure 5.1: The performance metrics for the recovery block detector, split across classes.

When breaking down the error detection of images per attack type, the results in Figure 5.1 are provided more context. This breakdown can be seen in Figure 5.2 and shows the F1 score for images based on the ground truth of those images and their attack type. As the attacks were broken down per type, ROC-AUC could not be used in these plots.[1] It should also be noted that due to the fact that the dataset is balanced, with 60 000 adversarial and 60 000 benign images, the benign column represents four times as many images as each of the per-attack columns.

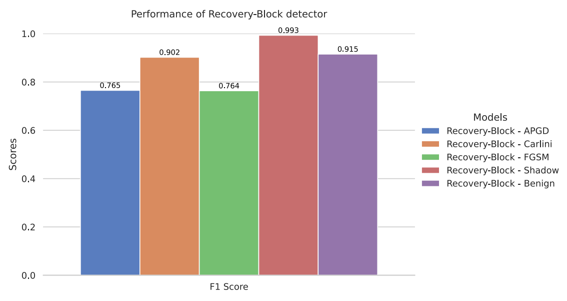
[1]ROC-AUC relies on mapping the true positive and false positive rate. The breakdown only contains true positives (or true negatives for benign images), which made calculating the false positive rate not possible. The implementation did not map the adversarial images back to their respective benign images. Thus, one would have to randomly select 15 000 images to represent the benign class to calculate the false positive rate. This could skew the results as a random selection of images could contain unknown biases, and thus it was decided not to use the ROC-AUC score.



(a) Spectral detector



(b) Binary detector



(c) Recovery Block detector

Figure 5.2: Figure breaking down the per-attack F1-score of the spectral and binary detector.

As can be seen, the distribution of detection is quite varied between the spectral and binary detectors. The binary detector is able to generalize adversarial noise across the four attacks but is only able to classify about 40% of the benign images correctly. On the other hand, the spectral detector performs well on errors triggered by adversarial images perturbed with Carlini&Wagner L2 and shadow attack. In terms of images with perturbations introduced by FGSM and APGD, it shows a lower level of coverage compared to the other two attacks. In order to understand this pattern for the spectral detector, Figure 5.3 breaks down each of these attacks in terms of how they perturb an image.

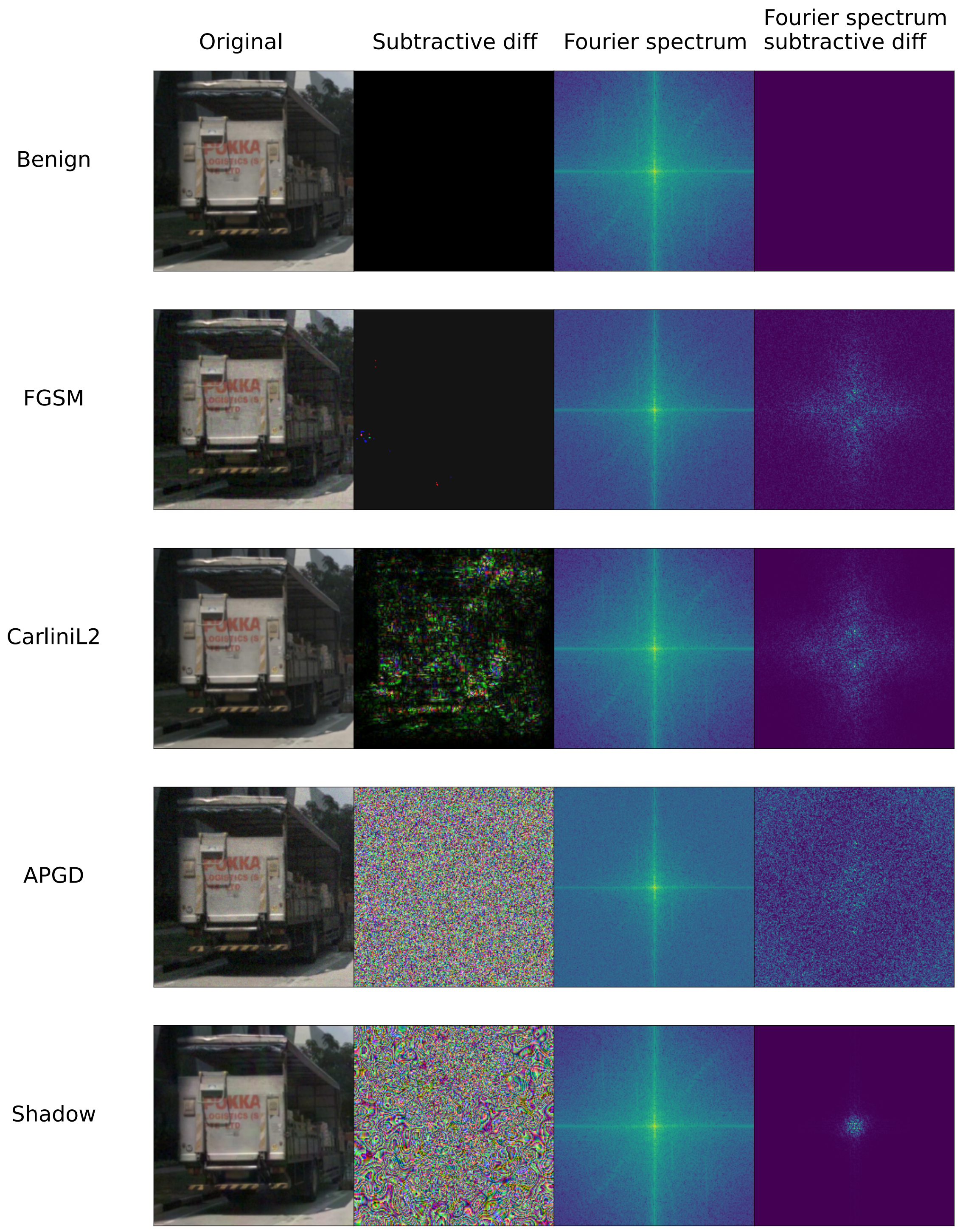


Figure 5.3: Figure showing the original image, subtractive difference, Fourier spectrum, and subtractive difference in the Fourier spectrum for images from each attack. Additionally the figure includes has the same for a benign image for reference. Large, high-quality versions of the images can also be seen in Appendix C

Figure 5.3 shows four different versions of an image from each attack in addition to the benign image. These four versions are the original image, the subtractive difference between the original image and the benign image, the Fourier spectrum of the image, and the subtractive difference of the Fourier spectrum to the benign image. The 'Original' column and the 'Fourier spectrum' column show the 'image' passed to the binary and spectral detector. The 'diff' columns display how the different attacks apply perturbations to an image from the perspective of the binary and spectral detector.

The images under 'subtractive diff' show that APGD and FGSM apply perturbations with relatively uniform sizes, while carliniL2 and shadow generally have a larger range of proportions. Additionally, APGD and FGSM appear to apply perturbations more in the mid to high-frequency spectrum compared to the Carlini and shadow attacks[2].

Figure 5.3 shows that the large variety in images that both the binary and spectral detectors had to detect and thus may explain why it was more difficult for the binary detector to generalize adversarial attack patterns. Where the spectral detector can rely on detecting bands of frequencies for detecting an attack, a binary detector would have to generalize for the different attack patterns and shapes, discerning them from natural image noise. As a result, it is likely that this could have made it difficult for the binary detector to consistently identify adversarial attacks, compared to the spectral detector.

As mentioned in section 2.3, disjunct error detection is critical to good redundancy systems. This means that both detectors would ideally ensure the correct detection of different sets of images. Figure 5.4 shows this distribution, where the two sets show the number of images correctly identified by the spectral detector and binary detector, acting as if they were not in the recovery block. As shown in the figure, both detectors had some overlap of correctly detected images. Following the architecture from McAllister et al. [28], the recovery block's implementation was in a 'blacklist'-fashion. This meant that both detectors needed to identify an image as adversarial before the recovery block detector classified it as adversarial. Based on this implementation, if the spectral detector incorrectly classifies an adversarial image as benign, it would not have been passed to the binary detector at all. This would have meant that the detectors could not have been used to their fullest capability. This would have meant a reduction in the total coverage and thus affected the performance of the recovery block as a whole. In terms of Figure 5.4, the figure shows the correctly detected images by both detectors, both benign and adversarial. The correctly identified benign images only need to be in one of the sets (red or green), while the adversarial images need to be in the overlapping set (yellow) to be correctly detected.

[2]FGSM's perturbations in the high-frequency Fourier spectrum is not that visible in Figure 5.3, but is more clear in the appendix images

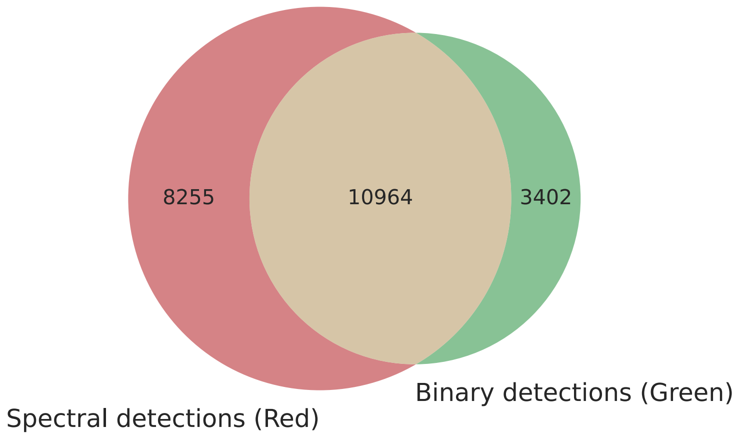


Figure 5.4: Venn diagram showing the overlap of correct classifications for the spectral and binary detectors in the recovery block. The yellow section indicates the correct detection overlap between the spectral and binary detectors.

5.2 N-Version programming detector

The utilization of the N-version programming detector is meant to build insights into how an architecture performs in relation to the diversity of the data. As mentioned, this consists of performing multiple transformations on the same image and comparing the performance of a model trained on these transformed outputs. Further, multiple detectors were individually trained on the four respective transformations to isolate the effects of any given transformation.

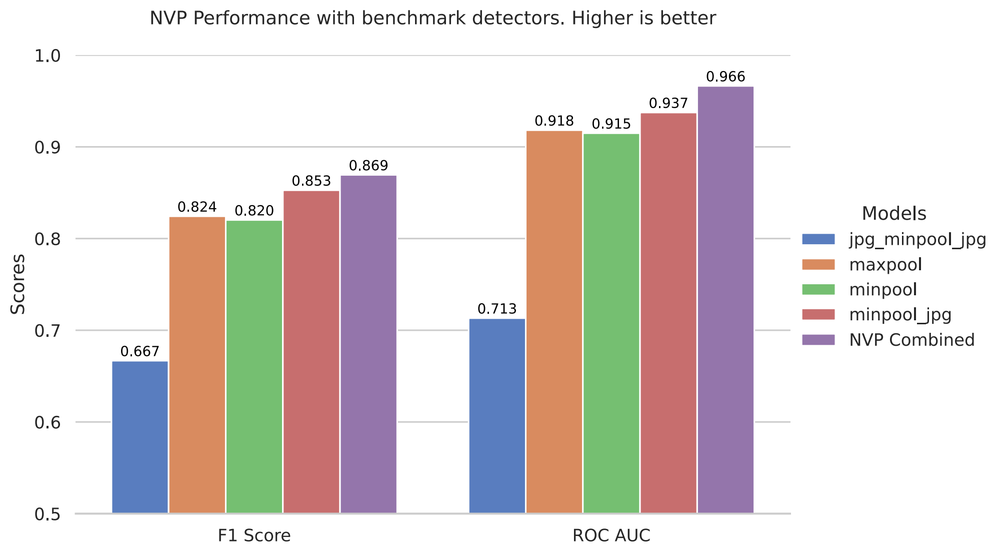
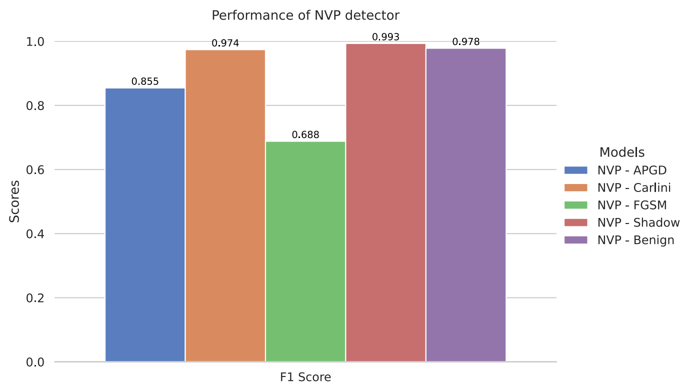


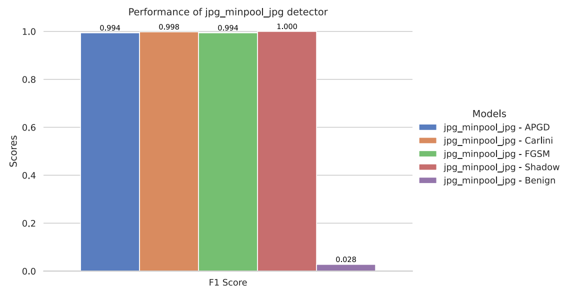
Figure 5.5: Performance of the combined NVP model compared to model only trained on each individual transformation. Note that the Y-scale starts at 0.5.

As shown in Figure 5.5, the NVP detector shows improvement over each of the individual benchmark detectors by some margin in both F1-score and ROC-AUC. However, it is important to note that despite this, all of the individual benchmark detectors performed well, considering that the detectors were trained on 1/4th the information provided to the NVP detector. This may have been due to the fact that the transformations were specifically selected due to their ability to induce detectable differences between benign and adversarial images.

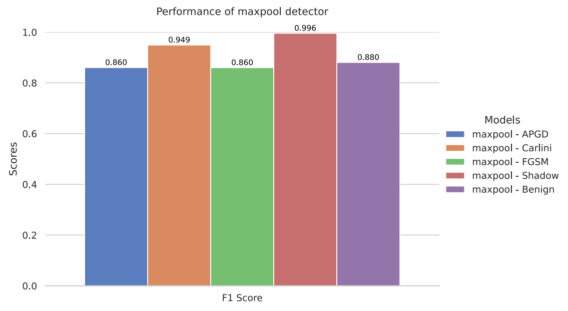
Within the individual detectors, a few things need to be noted. Out of the four transformations, the 'jpg_minpool_jpg' transformation performs quite differently in both F1-score and ROC-AUC. This is interesting, as both the 'minpool' and 'minpool_jpg' transformations performed quite well and used the same underlying functions as the 'jpg_minpool_jpg'.



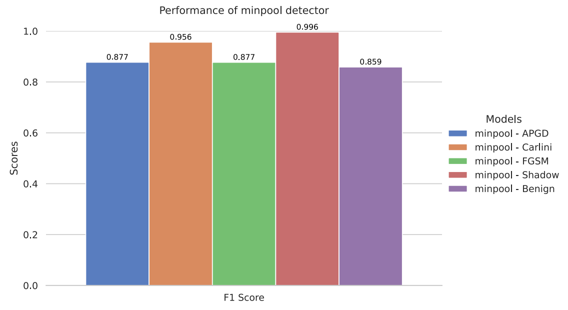
(a) NVP



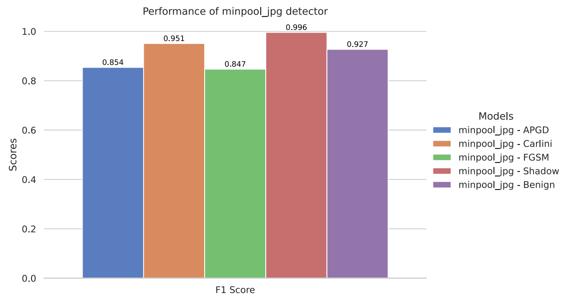
(b) 'jpg_minpool_jpg'



(c) 'maxpool'



(d) 'minpool'

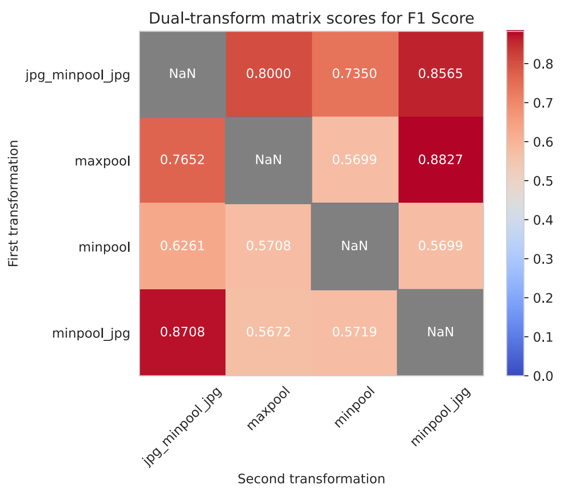


(e) 'minpool_jpg'

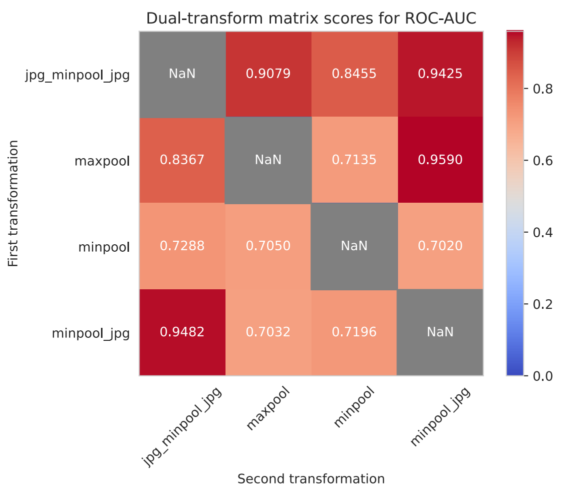
Figure 5.6: The per-attack F1-score for the NVP detector and the individual detectors.

Further insight into these can be seen in Figure 5.6. The plots show the F1-score per attack for each of the attacks as well as the benign images. As can be observed, most of the detectors are able to consistently correctly detect both Carlini&WagnerL2 and shadow attacks. Additionally, most detectors are also capable of detecting FGSM and APGD examples, albeit at a lower rate.

Two detectors that do not fit this pattern are the NVP and ‘jpg_minpool_jpg’ detectors. Similar to the binary detector, the ‘jpg_minpool_jpg’ detector generalizes well for the adversarial perturbations but is not able to differentiate them from natural image noise. The NVP detector, on the other hand, sacrifices performance in detecting FGSM for a significant improvement in detecting benign images.



(a) F1-score

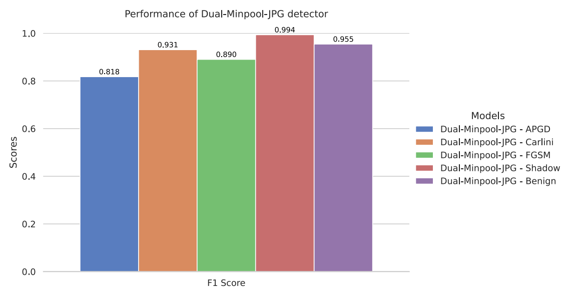


(b) ROC-AUC

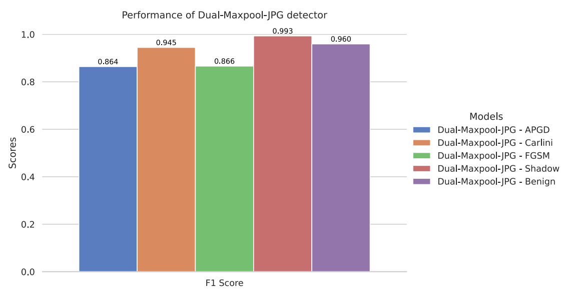
Figure 5.7: Performance of the dual-transformation detectors. The y-axis is the first transformation in the order, while the x-axis represents the second. Using the same transformation data for the first and second transformations does not make sense, so the diagonal is removed.

Additionally, several dual-transform detectors were trained. As recalled from section 4.7, these consisted of "NVP" detectors, but only had access to two input spaces, instead of four. Their performance can be seen in Figure 5.7. The X-axis indicates which transformation filled the first three channel-wise elements, and the Y-axis indicates the last three. As observed in the matrix, there was a large variance in performance. In most cases, the F1-score and ROC-AUC significantly degraded in performance, with a few notable exceptions. Comparing the results of the best-performing dual-transform detectors to the N-Version Programming detector shows that a doubling of input spaces can put the dual-transformers performance quite close to the performance of the NVP-detector.

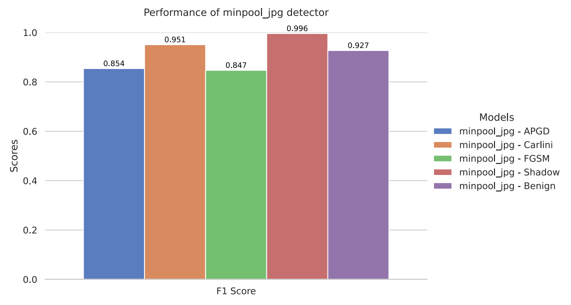
The top two performers were the ‘minpool_jpg’ + ‘jpg_minpool_jpg’ and the ‘maxpool’ + ‘minpool_jpg’ detectors, performing close to the NVP detector. Another thing to note is that the matrix is not mirrored across its diagonal, which may suggest that the channel order can also significantly impact a detector’s performance. One example is that the detector utilizing the ‘minpool_jpg’ as its first transformation and the ‘maxpool’ as its second performs significantly differently than the detector with this order of transformations reversed.



(a) ‘minpool_jpg’ + ‘jpg_minpool_jpg’



(b) ‘maxpool’ + ‘minpool_jpg’



(c) ‘minpool_jpg’. Copy of Figure 5.6 (e), added for convenience of reading.

Figure 5.8: Per-class detection distribution of dual-transform detectors with their respective transformations. It also contains the distribution for their joint individual detector ‘minpool_jpg’.

Breaking down the per-class performance of two best-performing dual-transform detectors in Figure 5.8 (a) and (b) shows that these two detectors improve very differently over their common individual detector ‘minpool_jpg’. The dual-transform with quite similar transformations, ‘minpool_jpg’ + ‘jpg_minpool_jpg’, improves error detection over the baseline for specific attacks. These improvements being more distinct for APGD and FGSM. The ‘maxpool’ + ‘minpool_jpg’ shows a more even distribution of per-class performance. This suggests that the diversity between transformations may be a factor affecting the distribution of coverage for attacks.

The dual-transform detectors in Figure 5.7 show that going from one to two transformations can drastically change performance, for better or worse. An example of the latter case is the utilization of a well-performing transformation such as ‘maxpool’ as its initial transformation, which, when combined with a diverse counterpart such as ‘minpool’ provides worse performance than their individual implementations.

Contrast this to the best performing but also the least data diverse dual-transformation detector; ‘minpool_jpg’ and ‘jpg_minpool_jpg’, shown in Figure 5.8. Comparing it to the more diverse dual-transformation detector ‘maxpool’ + ‘minpool_jpg’ points to the fact that data diversity, in general, is likely to improve the detection of adversarial attacks. Going back to the results in Figure 5.8, it must also be noted that how diverse these transformations are may dictate if the detection improvements come in specific classes or if the improvements are distributed across all classes.

Chapter 6

Discussion

This chapter will discuss the results presented in the previous chapter and tie them into the context of the research field. The first section will explain how the multi-model architecture performed and discuss important factors to consider for future works. The second section will discuss what the results show regarding data diversity and how an increase in input spaces can affect the model. The third section will present the general implications of architecture detectors and how they require additional considerations. The final section will discuss the limitations of this thesis and how future implementations can improve.

6.1 Multi-model architectures

The results from section 5.1 provide valuable insights. The large binary detector demonstrated a capacity to generalize attack perturbations but struggled to distinguish them from benign images. Large models such as the binary detector can be prone to improper generalization [75]. Despite the works by A. Kurakin et al. [71] and Madry et al. [23] denoting the ability of large models to show improved performance against adversarial attacks, it did not prove effective for the binary detector. Looking at the performance of the per-attack detection rate and the uniformity of perturbations introduced by each attack, it is not unlikely that the varied perturbation size in CarliniL2 and shadow attack played a part in them being more easily detected. Additionally, having these attacks operating in the low-frequency spectrum could have made the detector generalize for these perturbations instead.

A possible cause for the binary detector's performance is the lack of transformations. Adversarial attacks are generated with the purpose of being difficult to detect in the input space. As the detection operation is performed on the same input space, the operation will likely not be as effective. The additional parameter tuning to create even more difficult attacks would only have compounded this effect. With the works by Nathan Drenkow et al., [39] and Paula Harder et al. [38], showing the efficacy of detection with even simple transformations, it shows the importance of detecting adversarial attacks in an input space outside in which they were generated to be hidden. By observing the performance of any of the benchmark detectors from Figure 5.7, it can be observed that even a simple maxpool transformation can drastically improve the detection operation. While these two detectors are not strictly comparable, it points to the value of shifting the input space to enhance error detection.

From a fault tolerance perspective, this recovery block results bring up some interesting points. The disjunct sets in detection between the spectral detector and binary detector indicate that the existence of two operations can map to different domains given the same input space (image). However, the increased F1-score observed between the recovery block and spectral detector is not proportional to these disjunct sets in Figure 5.4. This highlights an important point regarding the applicability of sequential redundancy for detecting adversarial attacks and how architecture implementations can dictate coverage. This thesis' implementation of the recovery block was implemented with a 'blacklist'-configuration in accordance with the recovery block implementation from 'Handbook of Software Reliability Engineering'. This configuration requires both detectors to detect an image as adversarial before it is fully classified as adversarial. This also means that images detected as benign by the spectral detector, however wrong they may be, are not passed to the binary detector. This results in a lack of full utilization of the models, as only a subset of images is passed to the binary detector. Avizienis et al. [27] define this phenomenon as 'failure independence coverage', where one component's failure limits other components' ability to catch said failure. The 'blacklist'-configuration of the recovery block implementation would naturally favor the correct identification of benign images. This is most likely why, despite the binary detector's weak performance on benign images, the recovery block still has an increase in performance on benign images. The binary detector has likely corrected the spectral detector's detection of a benign image as adversarial.

The configuration of the fault-tolerant architecture is a trade-off between how one prioritizes the correct detection of benign and adversarial images. This thesis' implementation shows that this prioritization can have a notable effect on detection and the domains in which it is effective. As a result, it denotes the importance of evaluating per-component performance and how the component composition will affect the coverage of the detection architecture. In addition to architecture configurations that fully utilize the detection operations, the multi-model architectures also show the value of shifting the input space. This shift would increase the effectiveness of the detection operations. Future works should look into how the size of a model, with the same shifted input space, affects the detection operation and how different configurations of sequential redundancy architectures affect their coverage.

6.2 Data diversity

As mentioned in section 4.7, the N-Version Programming (NVP) detector was implemented to observe how an increase in input space would affect coverage. The results in section 5.2 show that adding more input states allows the detection operation to increase its domain. When broken down on a per-attack level, it can be observed that most of the individual detectors showed the same pattern of having more effective coverage on Carlini&WagnerL2 and shadow attacks than FGSM and APGD.

In order to understand this pattern, it may be useful to view it from the perspective of ensemble learning. In the ensemble learning field, the diversity within classifiers dictates performance and defines the distribution of this performance [76] [77]. In this field, a set of high-accuracy but low-variance classifiers is known to enhance the existing patterns of each individual detector [76]. Given the premise that the input spaces made it easier for the operation to detect Carlini&WagnerL2 and shadow attacks, the low diversity between the transformations would only have enhanced this pattern. It is interesting to see that when additional input spaces are added, APGD detection improves significantly compared to the other attacks. This may suggest that the APGD attacks are more similar to the other attacks than FGSM in this adversarial dataset.

The dual-transformation detectors also show that increased input spaces can also come with risk. Some detectors with certain combinations of input spaces have their coverage significantly reduced by the introduction of a secondary transformation. The fact that the order of transformations affected the results may provide some insight. As mentioned in section 4.7, a concern was that the models being pre-trained would make them prioritize the first three channels of an image, as they would only have pre-trained weights for those three channels. It was assumed that an increase would force the model to utilize the second transformation before stopping training. Efforts to prevent this by increasing the number of early stopping epochs did not work to the same degree as initially assumed.

However, this information may be useful in itself. Early layers in models are well known to capture low-level image features such as edges and/or color correlation, with later layers capturing more abstract and class-specific features [78] [79]. Recall from section 4.7 that adding a second transformation involved changing the initial input layer to accept a tensor with dimensions of 6x224x224. As the pre-trained weights only had pre-trained weights for the first three channels, it meant that the last three channels did not have any pre-trained weights. Thus, if the model performed well, it would mean that the initial layer found an optimal solution quickly. As early layers capture low-level image features, it is reasonable to conclude that the addition of an additional input space enhanced the dual-transform detectors' ability to utilize these low-level image features to detect adversarial attacks.

Another thing to note in Figure 5.6 is that the individual input spaces also greatly affect the domains in which to detect errors. An interesting contrast is how an initial JPEG compression stage separating the 'jpg_minpool_jpg' and the 'minpool_jpg' hampers the 'jpg_minpool_jpg' detectors ability to identify instances of benign images. This may be due to the fact that the use of JPEG compression also introduces compression artifacts which may be misconstrued as noise [80][81]. These compression artifacts, which may be imperceptible to humans, could perhaps be difficult to discern from natural noise after they have been compressed. This phenomenon is also supported by literature as attempts have been made to utilize JPG compression during pre-processing for the adversarial training of models attacked by FGSM [82] [83]. Similarly, it was noted that JPEG compression was ineffective for robustness training on low-resolution images, as the JPEG artifacts would confuse the model. While not explicitly related, it shows that JPEG compression on our relatively low-resolution dataset could make it difficult to discern adversarial perturbations and natural noise. However, when the input spaces for 'jpg_minpool_jpg' and 'minpool_jpg' are combined, they produce the best-performing dual-detector.

In terms of fault-tolerant architectures, this introduces some interesting insights into data diversity and input spaces. It shows that the individual performance of a detector is not necessarily predictive of a composition utilizing these detectors. Each composition of input spaces can change the domain of detected examples significantly. Similarly to the work by Nathan Drenkow et al. [39], the addition of more diverse images can allow for detectors to observe patterns for which only one transformed image could not. However, this diversity may come at the cost of reduced accuracy[76]. These results point to the applicability of fault-tolerant concepts in the adversarial image detection domain. That being said, it also denotes trade-offs and introduces dilemmas in which there are no correct answers. It could be interesting to address these issues more thoroughly in future works, such as how sufficiently trained detectors with diverse input spaces compare to homogeneous input spaces and how differently they perform.

6.3 Suitability of detection architectures

While not strictly tied to a research question, comparing the issues tied to the recovery block and the NVP architecture and the challenges sequential and parallel redundancy face in implementation is interesting. As noted in section 6.1, utilizing sequential redundancy relies heavily on disjunct error detection to be effective. For an optimal sequential redundancy architecture, the component detectors must be trained to detect errors that previous stages could not. As mentioned in section 6.1, this implementation was prone to a lack of 'failure independence coverage'. To mitigate this, a defender would have to somewhat predict errors that previous stages could not detect. Knowing this would ensure that the detectors were trained to ensure disjunct detection. However, it would require extensive testing to reliably know which domains a given detector would provide coverage for.

Looking at the performance of the parallel redundancy model, NVP also brings up a number of challenges. The NVP model showed that data diversity through increased input spaces is viable. However, it also showed that one input space is not necessarily indicative of the performance of an entire composition. Additionally, the diversity within a composition may affect the distribution of domains.

Both of these architectures show that as soon as adversarial attack detectors leave the space of single-stage 'robustness-based' detectors, fault-tolerance principles readily apply. When one introduces additional operations or input spaces, one suddenly needs to consider additional factors. Overlap in coverage and distribution of domains affecting results indicate that these additional factors can both amplify and undermine the detection of adversarial attacks. These factors having such a large influence call for future works to investigate methods in which to predict optimal configurations or compositions without requiring extensive run-time testing to discover them.

6.4 Limitations

It is important to understand the context in which the results and discussion are written. This section will present the known limitations to the work performed in the thesis and provide a scope in which to understand them.

6.4.1 Researcher-produced dataset

In terms of understanding the results, the limitations of the datasets are important to its context. As the dataset was entirely produced by the researcher, a risk of it being inadvertently skewed exists inherently in its production. The images and classes were deliberately selected to produce a strong performance on a ResNet-101 classifier and thus may contain unforeseen biases which may affect the performance on other tasks. As the nuScenes dataset is intended for benchmarking object detectors, the results must be considered within the context of the dataset.

Another known limitation is the dataset, as it could contain an imbalance in how well the classes are defined. As mentioned in chapter 4, all the images were selected to have above 64x64 pixel sizes. Additionally, the images were re-scaled to tensors in the shape of 224x224. As the images were taken from a vehicle, it is more than likely that the average image size of some classes was higher than others. Thus, classes such as 'vehicle' have more pixels for the PyTorch resizing transformation available to infer data and transform it into a 224x224 tensor. As mentioned previously, in subsection 4.4.1, a decrease in image size leads to more noise, leading to worse-trained classifiers. This disparity in image sizes could have affected classes that were further away, such as 'pedestrian' or 'inanimate,' making these images noisier. This could have made some attacks more effective, as some images would contain more natural noise for adversarial perturbations to ‘hide’ in.

Additionally, as noted during chapter 4, the image restrictions were reduced from 128 to 64 pixels in either dimension. However, this was done before the benign dataset was truncated to 150 000 images. Ideally, one would rather set a requirement of only needing 150 000 first and then see if a sufficient amount of images were above the 128-pixel threshold to saturate the need for 150 000 images in the dataset. Additionally, the script for truncating the dataset was random and did not select the largest images. In future implementations, it could be valuable to improve upon this process to ensure that the images were more uniform in size.

6.4.2 Diversity of adversarial examples

One major thing to note is the disparity of detection between the different attacks. As can be seen in Figure 5.2 and Figure 5.6, both the Carlini&WagnerL2 and Shadow attacks were consistently more easily detected by most detectors. This may point to the fact that some attacks were easier to detect; thus, the detectors were more likely to optimize for detecting those attacks. This may have skewed the detectors’ ability to detect all attacks, accepting a sub-optimal point where most attacks were detected but not all. As most detectors show a similar pattern, it may point to the fact that the generated Carlini&WagnerL2 and Shadow were weaker than their APGD and FGSM counterparts. This may have been caused by an adjustment of the attack parameters to produce challenging images. Comparing how these adjustments shifted the ‘difficulty’ of each attack is difficult to determine. As a result, it may have inadvertently made FGSM and APGD attacks significantly more difficult relative to how difficult the other attacks were. To mitigate this in the future, additional testing should be done to ensure that the selected attack parameters create images that trick classifiers to the same degree as all the other attacks.

Another limitation of the adversarial dataset is that each attack was only generated with one set of parameters. One example is attacks such as APGD, where the images are perturbed with only one set of epsilons. Utilizing the same set of parameters for each of the attacks could have made it much easier to generalize for the adversarial perturbations introduced by the attacks, as all four attacks would attack an image in the same manner every time. While the utilization of four attacks should have mitigated this risk, it is difficult to exclude the fact that it may have made it easier to generalize the noise introduced by the attacks. As a result, the ability of complex models to easily generalize for adversarial noise across multiple attacks should not be taken as a definitive assertion. Rather, it should be taken as an indication, available for further investigation in future works.

6.4.3 Detector training

Regarding the training of the detectors, it is important to note that the models were trained with the intention of acting as *representative* models, as it would allow for seeing how the fault-tolerant architectures affected the detection performance over the component detectors. As mentioned in chapter 4, steps were taken to ensure that the models would be reasonably well-trained. However, several additional measures would have to be made if one would attempt to conclude that these models were optimally trained. While the heuristic selection of hyperparameters and early stopping methods should be sufficient for representative models, additional steps could have been taken to improve model training. These include but are not limited to Bayesian search of hyperparameters for shallow models [84] or learning rate schedulers for deep models [85]. Additionally, the individual NVP models and dual-transform models were trained with the same parameters instead of a different set of parameters for all. This was done due to time constraints. This means that while the results are likely representative of the fault-tolerant architectures, they should not be considered optimal models, and further optimizations need to be applied, relative to the detectors operating environment.

Chapter 7

Conclusion

This thesis has looked at the concept of how fault-tolerant architectures can be implemented and using these architectures to create generalizable insights into how these architectures should be implemented in the future. Specifically, this thesis has looked at how multi-model and design diverse data architectures can support the goal of detecting adversarial examples. Focusing on the concept of increasing the input space and operations used to produce detection domains, this thesis implemented the fault-tolerant detection architectures recovery block and N-Version programming. Using a modified version of the nuScenes dataset, an adversarial dataset was generated based on four attacks; Fast Gradient Sign Method, Auto-Projected Gradient Descent, Carlini&Wagner L2, and Shadow attack. The two architectures were trained on this dataset, using a heuristic selection of hyperparameters and performing early stopping when the model did not improve on the validation set.

Upon testing the recovery block, it was clear that an increase in detection operations was beneficial to increase coverage. However, implementation did not necessarily support the complete utilization of each component detector. The NVP detector, its individual detectors, and dual-transform detectors showed that an increase in data diversity could increase the detection of adversarial attacks, but the transformations need to be selected carefully. The composition of these transformations can skew how this improved detection rate is distributed among attacks, and a subpar selection of transformations can significantly hamper performance. Both detectors show that the fault-tolerant concepts of coverage, design diversity, and redundancy are transferable to the domain of detecting adversarial attacks. However, when implementing fault-tolerant architectures to detect adversarial attacks, careful consideration needs to be made in terms composition of detectors and how the architecture supports them. This thesis shows that treating adversarial detection as a fault-tolerance problem is viable, but finding optimal configurations and compositions is difficult. In future works, it may be interesting to look into methods for predicting these optimal compositions without the requirement of exhaustive testing.

Bibliography

[1] European Union Agency for Cybersecurity et al. *ENISA threat landscape 2022 : July 2021 to July 2022*. Ed. by R Svetozarov Naydenov et al. 2022. DOI: doi/10.2824/764318.

[2] Tom B Brown and Catherine Olsson. *Introducing the unrestricted adversarial examples challenge*. Sept. 2018. URL: https://ai.googleblog.com/2018/09/introducing-unrestricted-adversarial.html.

[3] Ann Johnson Ram Shankar Siva Kumar. *Cyberattacks against machine learning systems are more common than you think*. Mar. 2021. URL: https://www.microsoft.com/en-us/security/blog/2020/10/22/cyberattacks-against-machine-learning-systems-are-more-common-than-you-think/.

[4] Patrick Moen Allport. "Detection of adversarial attacks in computer vision: A literature review". Unpublished work from course IT3915 Master in Informatics, Preparatory Project. Supervisor: Montecchi, Leonardo. Nov. 2022.

[5] Ruoxi Chen et al. "Salient feature extractor for adversarial defense on Deep Neural Networks". In: *Information Sciences* 600 (2022), pp. 118–143. DOI: 10.1016/j.ins.2022.03.056.

[6] Weiqi Fan et al. "Hybrid defense for deep neural networks: An integration of detecting and cleaning adversarial perturbations". In: *2019 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)* (2019). DOI: 10.1109/icmew.2019.00-85.

[7] Victor Wiley and Thomas Lucas. "Computer Vision and image processing: A paper review". In: *International Journal of Artificial Intelligence Research* 2.1 (2018), p. 22. DOI: 10.29099/ijair.v2i1.42.

[8] Supriya V. Mahadevkar et al. "A Review on Machine Learning Styles in Computer Vision—Techniques and Future Directions". In: *IEEE Access* 10 (2022), pp. 107293–107329. DOI: 10.1109/ACCESS.2022.3209825.

[9] Oscar Cosido et al. "Hybridization of Convergent Photogrammetry, Computer Vision, and Artificial Intelligence for Digital Documentation of Cultural Heritage - A Case Study: The Magdalena Palace". In: *2014 International Conference on Cyberworlds*. 2014, pp. 369–376. DOI: `10.1109/CW.2014.58`.

[10] Arthur Francisco Fernandes, João Ricardo Dórea, and Guilherme Jordão Rosa. "Image Analysis and Computer Vision Applications in Animal Sciences: An overview". In: *Frontiers in Veterinary Science* 7 (2020). DOI: `10.3389/fvets.2020.551269`.

[11] Da-Hai Xia et al. "Review-material degradation assessed by digital image processing: Fundamentals, progresses, and challenges". In: *Journal of Materials Science and Technology* 53 (2020), pp. 146–162. DOI: `10.1016/j.jmst.2020.04.033`.

[12] Keumsun Park, Minah Chae, and Jae Hyuk Cho. "Image pre-processing method of machine learning for EDGE detection with Image Signal Processor Enhancement". In: *Micromachines* 12.1 (2021), p. 73. DOI: `10.3390/mi12010073`.

[13] Laith Alzubaidi et al. "Review of Deep Learning: Concepts, CNN architectures, challenges, applications, future directions". In: *Journal of Big Data* 8.1 (2021). DOI: `10.1186/s40537-021-00444-8`.

[14] Junyi Chai et al. "Deep Learning in Computer Vision: A critical review of emerging techniques and application scenarios". In: *Machine Learning with Applications* 6 (2021), p. 100134. DOI: `10.1016/j.mlwa.2021.100134`.

[15] Joseph Redmon et al. "You only look once: Unified, real-time object detection". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016). DOI: `10.1109/cvpr.2016.91`.

[16] Joseph Redmon et al. "You only look once: Unified, real-time object detection". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016). DOI: `10.1109/cvpr.2016.91`.

[17] I J Goodfellow, J Shlens, and C Szegedy. In: *Explaining and harnessing adversarial examples* (2014). DOI: `https://doi.org/10.48550/arXiv.1412.6572`.

[18] Hongshuo Liang et al. "Adversarial attack and defense: A survey". In: *Electronics* 11.8 (2022), p. 1283. DOI: `10.3390/electronics11081283`.

[19] Naveed Akhtar et al. "Advances in Adversarial Attacks and Defenses in Computer Vision: A Survey". In: *IEEE Access* 9 (2021), pp. 155161–155196. DOI: `10.1109/ACCESS.2021.3127960`.

[20] Nicolas Papernot et al. "Distillation as a defense to adversarial perturbations against Deep Neural Networks". In: *2016 IEEE Symposium on Security and Privacy (SP)* (2016). DOI: `10.1109/sp.2016.41`.

[21] Anish Athalye, Nicholas Carlini, and David Wagner. *Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples*. July 2018. URL: `https://arxiv.org/abs/1802.00420`.

[22] Reuben Feinman et al. *Detecting adversarial samples from artifacts*. Nov. 2017. URL: `https://arxiv.org/abs/1703.00410`.

[23] Aleksander Madry et al. "Towards deep learning models resistant to adversarial attacks". In: Cited by: 2139. 2018. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85083954061&partnerID=40&md5=84bf66031966d7b8f24a2260e59ff64c.

[24] Han Xu et al. "Adversarial attacks and defenses in images, graphs and text: A Review". In: *International Journal of Automation and Computing* 17.2 (2020), pp. 151–178. DOI: 10.1007/s11633-019-1211-x.

[25] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and harnessing adversarial examples*. Jan. 1970. URL: https://research.google/pubs/pub43405/.

[26] Nicholas Carlini and David Wagner. "Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods". In: *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. AISec '17. Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 3–14. ISBN: 9781450352024. DOI: 10.1145/3128572.3140444. URL: https://doi.org/10.1145/3128572.3140444.

[27] A. Avizienis et al. "Basic concepts and taxonomy of dependable and secure computing". In: *IEEE Transactions on Dependable and Secure Computing* 1.1 (2004), pp. 11–33. DOI: 10.1109/tdsc.2004.2.

[28] D. F. McAllister and M. A. Vouk. *Handbook of Software Reliability Engineering*. Ed. by M. R. Lyu. McGraw-Hill, 1996. Chap. 14.

[29] URL: https://www.scopus.com/results/citedbyresults.uri?sort=plf-f&amp;cite=2-s2.0-84929471545&amp;src=s&amp;imp=t&amp;sot=cite&amp;sdt=a&amp;sl=0&amp;origin=inward&amp;editSaveSearch=&amp;txGid=4e0ce9409ad79763ae9e206d30d62437.

[30] URL: https://dl.acm.org/doi/book/10.5555/239425.

[31] "ISO/IEC/IEEE International Standard - Systems and software engineering–Vocabulary". In: *ISO/IEC/IEEE 24765:2017(E)* (2017), pp. 1–541. DOI: 10.1109/IEEESTD.2017.8016712.

[32] Christian Szegedy et al. "Intriguing properties of neural networks". In: (Dec. 2013).

[33] Alexander Bastounis, Anders C Hansen, and Verner Vlačić. *The mathematics of adversarial attacks in AI – why deep learning is unstable despite the existence of stable neural networks*. Aug. 2021. URL: https://arxiv.org/abs/2109.06098v1.

[34] A. M. Turing. "On computable numbers, with an application to the Entscheidungsproblem". In: *Proceedings of the London Mathematical Society* s2-42.1 (1937), pp. 230–265. DOI: 10.1112/plms/s2-42.1.230.

[35] Xiaoyu Zhang et al. "Cassandra: Detecting Trojaned Networks From Adversarial Perturbations". In: *IEEE Access* 9 (2021), pp. 135856–135867. DOI: 10.1109/ACCESS.2021.3101289.

[36] Avizienis and Kelly. "Fault Tolerance by Design Diversity: Concepts and Experiments". In: *Computer* 17.8 (1984), pp. 67–80. DOI: 10.1109/MC.1984.1659219.

[37] J. Musa and G. Fuoco. *Handbook of Software Reliability Engineering.* Ed. by M. R. Lyu. McGraw-Hill, 1996. Chap. 5.

[38] Paula Harder et al. "SpectralDefense: Detecting Adversarial Attacks on CNNs in the Fourier Domain". In: *2021 International Joint Conference on Neural Networks (IJCNN)* (2021), pp. 1–8. DOI: `10.48550/arXiv.2103.03000`.

[39] Nathan Drenkow, Neil Fendley, and Philippe Burlina. "Attack agnostic detection of adversarial examples via random subspace analysis". In: *2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)* (2022). DOI: `10.1109/wacv51458.2022.00287`.

[40] Shuo Wang et al. "Adversarial detection by latent style transformations". In: *IEEE Transactions on Information Forensics and Security* 17 (2022), pp. 1099–1114. DOI: `10.1109/tifs.2022.3155975`.

[41] Karren Yang et al. "Defending multimodal fusion models against single-source adversaries". In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021). DOI: `10.1109/cvpr46437.2021.00335`.

[42] A Geiger et al. "Vision Meets Robotics: The kitti dataset". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1231–1237. DOI: `10.1177/0278364913491297`.

[43] G. van Rossum. *Python tutorial.* Tech. rep. CS-R9526. Amsterdam: Centrum voor Wiskunde en Informatica (CWI), May 1995.

[44] Django Software Foundation. *Django.* Version 2.2. May 22, 2023. URL: `https://djangoproject.com`.

[45] Miguel Grinberg. *Flask web development: developing web applications with python.* " O'Reilly Media, Inc.", 2018.

[46] *Python Package Index - PyPI.* URL: `https://pypi.org/` (visited on 05/22/2023).

[47] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32.* Curran Associates, Inc., 2019, pp. 8024–8035.

[48] *Pytorch Ecosystem tools.* URL: `https://pytorch.org/ecosystem/` (visited on 05/22/2023).

[49] William Falcon et al. *PyTorchLightning/pytorch-lightning: 0.7.6 release.* Version 0.7.6. May 2020. DOI: `10.5281/zenodo.3828935`. URL: `https://doi.org/10.5281/zenodo.3828935`.

[50] URL: `https://www.pytorchlightning.ai/team`.

[51] Maria-Irina Nicolae et al. *Adversarial robustness toolbox v1.0.0.* Nov. 2019. URL: `https://arxiv.org/abs/1807.01069`.

[52] *Idun.* URL: `https://www.hpc.ntnu.no/idun/` (visited on 05/22/2023).

[53] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444. DOI: `10.1038/nature14539`.

[54] Alon Halevy, Peter Norvig, and Fernando Pereira. "The unreasonable effectiveness of data". In: *IEEE Intelligent Systems* 24.2 (2009), pp. 8–12. DOI: `10.1109/mis.2009.36`.

[55] Ekim Yurtsever et al. "A survey of autonomous driving: common practices and emerging technologies". In: *IEEE Access* 8 (2020), pp. 58443–58469. DOI: `10.1109/access.2020.2983149`.

[56] Holger Caesar et al. "nuScenes: A multimodal dataset for autonomous driving". In: *CVPR*. 2020.

[57] *nuScenes detection task*. URL: `https://www.nuscenes.org/object-detection?externalData=all&mapData=all&modalities=Any` (visited on 05/22/2023).

[58] Bolei Zhou et al. "Learning deep features for discriminative localization". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016). DOI: `10.1109/cvpr.2016.319`.

[59] Shifeng Zhang et al. "Occlusion-aware R-CNN: Detecting pedestrians in a crowd". In: *Computer Vision – ECCV 2018* (2018), pp. 657–674. DOI: `10.1007/978-3-030-01219-9_39`.

[60] Feng Cen and Guanghui Wang. "Boosting Occluded Image Classification via Subspace Decomposition-Based Estimation of Deep Features". In: *IEEE Transactions on Cybernetics* 50.7 (2020), pp. 3409–3422. DOI: `10.1109/TCYB.2019.2931067`.

[61] Ajay Kumar Singh et al. "Wavelet based histogram of oriented gradients feature descriptors for classification of partially occluded objects". In: *International Journal of Intelligent Systems and Applications* 7.3 (2015), pp. 54–61. DOI: `10.5815/ijisa.2015.03.07`.

[62] Zhihao Wang, Jian Chen, and Steven C. H. Hoi. "Deep Learning for Image Super-Resolution: A Survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.10 (2021), pp. 3365–3387. DOI: `10.1109/TPAMI.2020.2982166`.

[63] Yanting Pei et al. "Effects of image degradation and degradation removal to CNN-based image classification". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.4 (2021), pp. 1239–1253. DOI: `10.1109/tpami.2019.2950923`.

[64] Kaiming He et al. "Deep residual learning for image recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016). DOI: `10.1109/cvpr.2016.90`.

[65] Justin M. Johnson and Taghi M. Khoshgoftaar. "Survey on deep learning with class imbalance". In: *Journal of Big Data* 6.1 (2019). DOI: `10.1186/s40537-019-0192-5`.

[66] Nicholas Carlini and David Wagner. "Towards evaluating the robustness of neural networks". In: *2017 IEEE Symposium on Security and Privacy (SP)* (2017). DOI: `10.1109/sp.2017.49`.

[67] Francesco Croce and Matthias Hein. *Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks*. Aug. 2020. URL: `https://arxiv.org/abs/2003.01690`.

[68] Amin Ghiasi, Ali Shafahi, and Tom Goldstein. *Breaking certified defenses: Semantic adversarial examples with spoofed robustness certificates*. Mar. 2020. URL: `https://arxiv.org/abs/2003.08937`.

[69] *ART Attacks*. URL: `https://github.com/Trusted-AI/adversarial-robustness-toolbox/wiki/ART-Attacks`.

[70] Naveed Akhtar and Ajmal Mian. "Threat of adversarial attacks on Deep Learning in Computer Vision: A survey". In: *IEEE Access* 6 (2018), pp. 14410–14430. DOI: `10.1109/access.2018.2807385`.

[71] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. "Adversarial machine learning at scale". In: Cited by: 631. 2017. URL: `https://www.scopus.com/inward/record.uri?eid=2-s2.0-85088231002&partnerID=40&md5=3817712882165d26bde2bb06a4f58137`.

[72] Garvesh Raskutti, Martin J. Wainwright, and Bin Yu. "Early stopping for non-parametric regression: An optimal data-dependent stopping rule". In: *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)* (2011). DOI: `10.1109/allerton.2011.6120320`.

[73] Rohan Reddy Mekala Fraunhofer USA CESE et al. "Metamorphic filtering of black-box adversarial attacks on multi-network face recognition models". In: *ACM Conferences* (June 2020). DOI: `10.1145/3387940.3391483`.

[74] Andrew P. Bradley. "The use of the area under the ROC curve in the evaluation of machine learning algorithms". In: *Pattern Recognition* 30.7 (1997). Cited by: 4493; All Open Access, Green Open Access, pp. 1145–1159. DOI: `10.1016/S0031-3203(96)00142-2`. URL: `https://www.scopus.com/inward/record.uri?eid=2-s2.0-0031191630&doi=10.1016%2fS0031-3203%2896%2900142-2&partnerID=40&md5=c3e22645a8533680341c9d1719287600`.

[75] Lutz Prechelt. "Early stopping — but when?" In: *Lecture Notes in Computer Science* (2012), pp. 53–67. DOI: `10.1007/978-3-642-35289-8_5`.

[76] Shuo Wang and Xin Yao. "Diversity analysis on imbalanced data sets by using ensemble models". In: *2009 IEEE Symposium on Computational Intelligence and Data Mining*. 2009, pp. 324–331. DOI: `10.1109/CIDM.2009.4938667`.

[77] Gavin Brown, Jeremy L. Wyatt, and Peter Tiňo. "Managing diversity in regression ensembles". In: *Journal of Machine Learning Research* 6 (2005). Cited by: 262. URL: `https://www.scopus.com/inward/record.uri?eid=2-s2.0-25444484657&partnerID=40&md5=b41c4626de5eb1b4077b7dea16d484de`.

[78] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[79] Matthew D. Zeiler and Rob Fergus. "Visualizing and understanding Convolutional Networks". In: *Computer Vision – ECCV 2014* (2014), pp. 818–833. DOI: `10.1007/978-3-319-10590-1_53`.

[80] Bin Li et al. "Revealing the trace of high-quality JPEG compression through quantization noise analysis". In: *IEEE Transactions on Information Forensics and Security* 10.3 (2015), pp. 558–573. DOI: `10.1109/tifs.2015.2389148`.

[81] Bruce K. Ho et al. "Mathematical model to quantify JPEG block artifacts". In: *SPIE Proceedings* (1993). DOI: `10.1117/12.146974`.

[82] Gintare Karolina Dziugaite, Zoubin Ghahramani, and Daniel M. Roy. *A study of the effect of JPG compression on adversarial images*. Aug. 2016. URL: `https://arxiv.org/abs/1608.00853`.

[83] Nilaksh Das et al. *Keeping the bad guys out: Protecting and vaccinating deep learning with JPEG compression*. May 2017. URL: `https://arxiv.org/abs/1705.02900`.

[84] Alma Rahat and Michael Wood. "On bayesian search for the feasible space under computationally expensive constraints". In: *Machine Learning, Optimization, and Data Science* (2020), pp. 529–540. DOI: `10.1007/978-3-030-64580-9_44`.

[85] Stéphane d'Ascoli, Maria Refinetti, and Giulio Biroli. *Optimal learning rate schedules in high-dimensional non-convex optimization problems*. Feb. 2022. URL: `https://arxiv.org/abs/2202.04509`.

Glossary

Adversarial Example (AE) A singular instance of a perturbed image used in an adversarial attack. 6, 7, 8, 9, 11, 18, 23, 36, 39, 43, 69

Adversarial Attack (AA) An attack consisting of one or more images with perturbations designed avoid, evade or otherwise disrupt proper clasification of one or more objects. i, 2, 4, 6, 7, 8, 9, 11, 12, 16, 20, 26, 37

COCO A data-format for describing bounding boxes. Defines bounding boxes based on x and y coordinates of the bounding box and defining the width and height. Documentation here. 83

Discrete Fourier Transform (DFT) A discrete version of the Fourier Transform used on digital data to map input values into a discrete range in the frequency domain. 17, 43, 45

Generative Adversarial Network (GAN) A class of machine learning frameworks playing a zero-sum game consisting of a generator, which attempts to produce adversarial examples, and a discriminator whom attempts to detect them. In such a network, both become iteratively better at detecting and generating examples until they reach an equilibrium . 17, 19

Machine Learning (ML) A field referring to a model focused machine-system in which one trains and adapts said model enhance a specific task. 8

Natural Examples Images in the dataset which have not had any data augmentation transformation applied to them.. 43

PASCAL-VOC A data-format for describing bounding boxes. Defines bounding boxes by the four corners of the bounding box. Documentation here. 83

Systematic Literature Review (SLR) A systematic literature review is a paper review which looks at a number of papers from a research field in order to point out research gaps, synthesise new information and produce a research position. 2, 3, 9, 16, 17, 18, 20, 22

Appendices

Appendix A

Method

Original class	**Mapped class in 4-class dataset**
animal	pedestrian
human.pedestrian.adult	pedestrian
human.pedestrian.child	pedestrian
human.pedestrian.construction_worker	pedestrian
human.pedestrian.personal_mobility	pedestrian
human.pedestrian.police_officer	pedestrian
human.pedestrian.stroller	pedestrian
human.pedestrian.wheelchair	pedestrian
movable_object.barrier	static_heavy
movable_object.debris	static_heavy
movable_object.pushable_pullable	static_light
movable_object.trafficcone	static_light
static_object.bicycle_rack	static_heavy
vehicle.bicycle	vehicle
vehicle.bus.bendy	vehicle
vehicle.bus.rigid	vehicle
vehicle.car	vehicle
vehicle.construction	vehicle
vehicle.emergency.ambulance	vehicle
vehicle.emergency.police	vehicle
vehicle.motorcycle	vehicle
vehicle.trailer	vehicle
vehicle.truck	vehicle

Table A.1: Table showing the mapping between the original 23 classes and their new class in the 4-class dataset.

Original class	**Mapped class in 3-class dataset**
animal	pedestrian
human.pedestrian.adult	pedestrian
human.pedestrian.child	pedestrian
human.pedestrian.construction_worker	pedestrian
human.pedestrian.personal_mobility	pedestrian
human.pedestrian.police_officer	pedestrian
human.pedestrian.stroller	pedestrian
human.pedestrian.wheelchair	pedestrian
movable_object.barrier	inanimate
movable_object.debris	inanimate
movable_object.pushable_pullable	inanimate
movable_object.trafficcone	inanimate
static_object.bicycle_rack	inanimate
vehicle.bicycle	vehicle
vehicle.bus.bendy	vehicle
vehicle.bus.rigid	vehicle
vehicle.car	vehicle
vehicle.construction	vehicle
vehicle.emergency.ambulance	vehicle
vehicle.emergency.police	vehicle
vehicle.motorcycle	vehicle
vehicle.trailer	vehicle
vehicle.truck	vehicle

Table A.2: Table showing the mapping between the original 23 classes and their new class in the 3-class dataset.

Appendix B

Pseudocode

Algorithm 1: Pseudocode of generation of customized dataset

```
1  nuscenes_dataset = download_nuscenes_dataset();
2  2d_annotations = nuscenes.export_2d_annotations_as_json(nuscenes_dataset);
   /* This file location represents a storage location on disk          */
3  cropped_file_location = get_location_on_disk();
4  for annotation in 2d_annotations do
5      visibility = annotation.get_visibility();
6      if visibility >= 60% then
7          image = annotation.get_image();
           /* Coordinates were in COCO-format and need to be converted to
              the PASCAL-VOC format                                        */
8          mid_x, mid_y, bbox_width, bbox_height = annotation.bbox;
9          x_min, y_min, x_max, y_max =
10             convert_coco_to_Pascal_VOC(mid_x, mid_y, bbox_width,
                bbox_height);
11         cropped_image = image.crop(x_min, y_min, x_max, y_max);
12         cropped_image.save(cropped_file_location);
13 cropped_images = load_files(cropped_file_location);
   /* This file location represents a storage location on disk          */
14 filtered_file_location = get_location_on_disk();
15 for image in cropped_images do
16     width, height = image.get_width_height();
17     if width >= 64 and height >= 64 then
18         class = image.class;
19         image.move(filtered_file_locationclass);
```

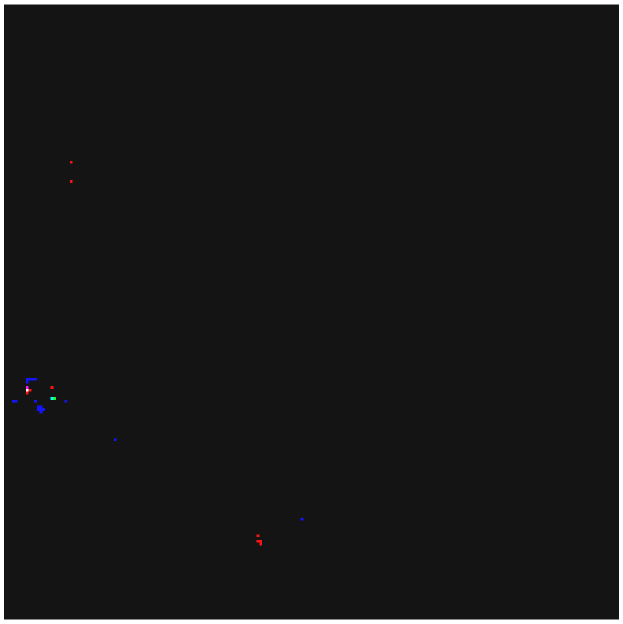
Algorithm 2: Pseudocode of generation of adversarial dataset

```
   /* This is the location of the benign images                          */
1  filtered_file_location = get_location_on_disk();
2  benign_dataset = load_dataset(filtered_file_location);
   /* This samples a representative subset of the benign images, 10% of
      the original size.                                                 */
3  sampled_benign_dataset = sample(benign_dataset, 10);
4  attacks = [APGD(), Carlini(), FGSM(), Shadow()];
5  for image,adv_save_location in sampled_benign_dataset do
6      for attack in attacks do
7          adversarial_image = attack.generate(image);
           /* Clips potential image values outside of the valid range of
              0,1                                                        */
8          clipped = adversarial_image.clip(0,1);
           /* OpenCV requires images to be in the BGR format.            */
9          bgr_clipped = clipped.color_format('BGR');
10         bgr_clipped.save_as_tiff(adv_save_location);
```

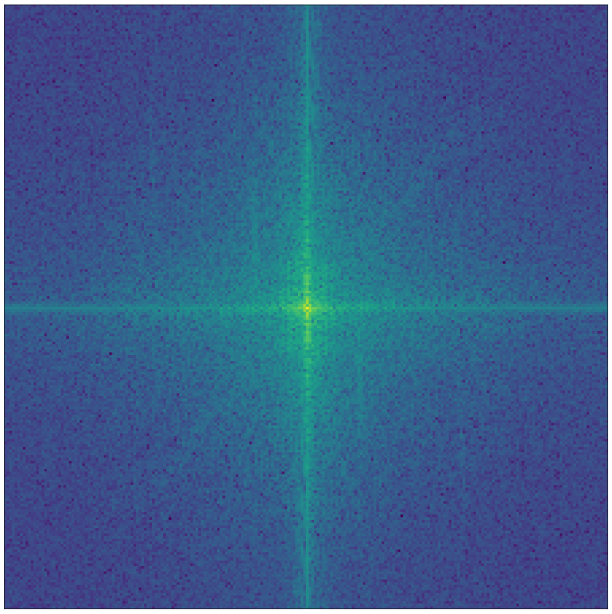
Appendix C

Results

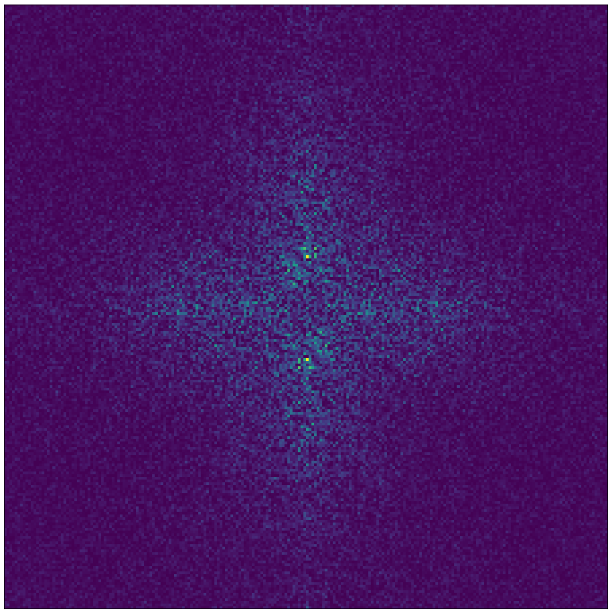
(a) Original



(b) Subtractive diff



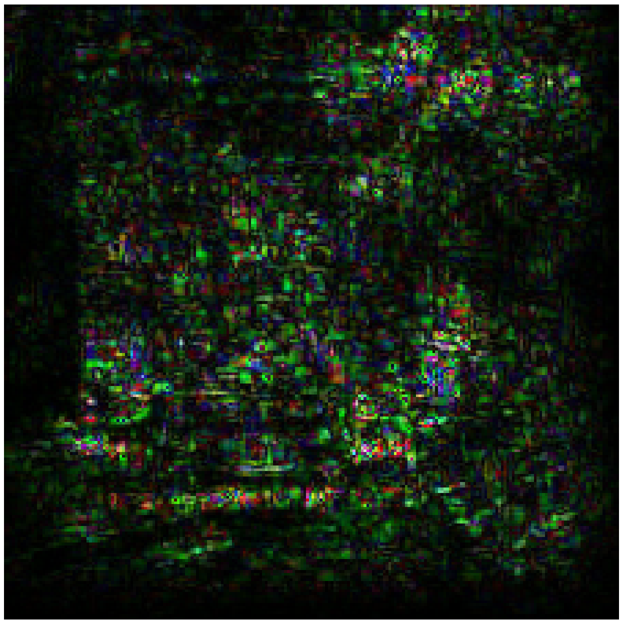
(c) Fourier spectrum



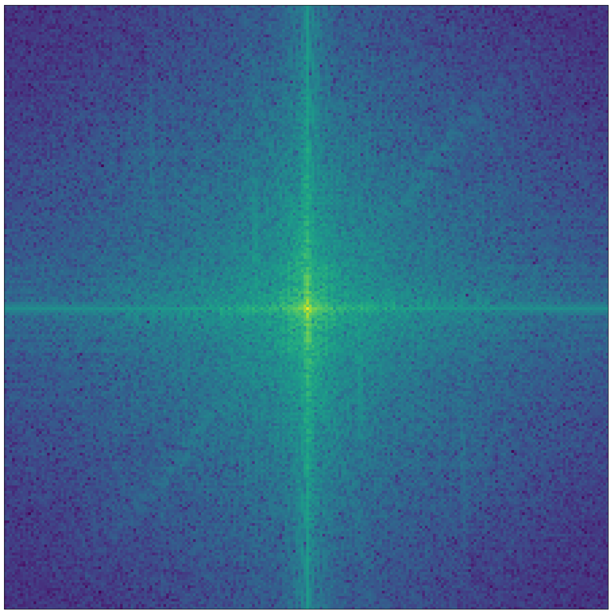
(d) Fourier spectrum Subtractive diff

Figure C.1: Original, Subtractive difference, Fourier spectrum and Subtractive Fourier spectrum for FGSM images.

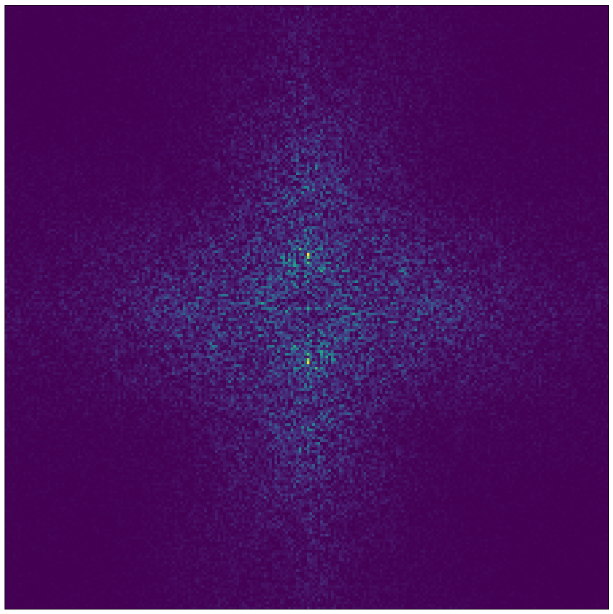
(a) Original



(b) Subtractive diff



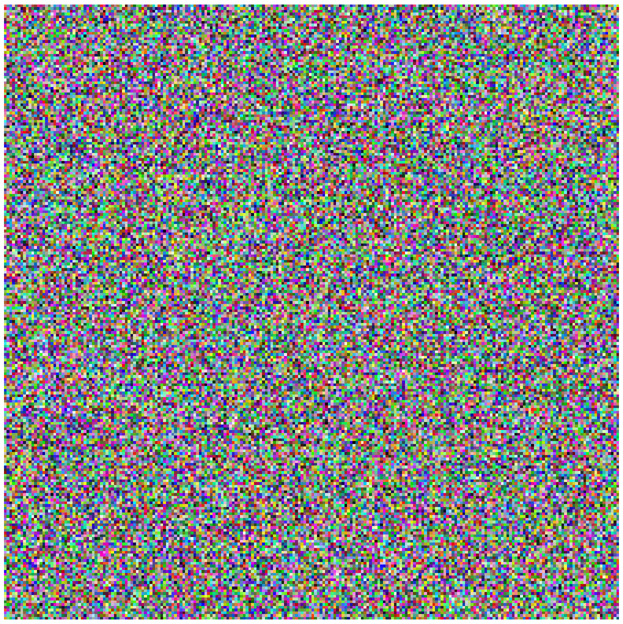
(c) Fourier spectrum



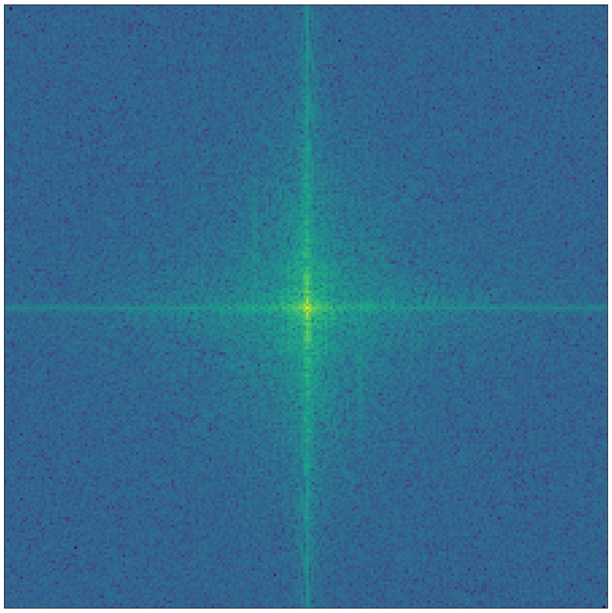
(d) Fourier spectrum Subtractive diff

Figure C.2: Original, Subtractive difference, Fourier spectrum and Subtractive fourier spectrum for carlini images.

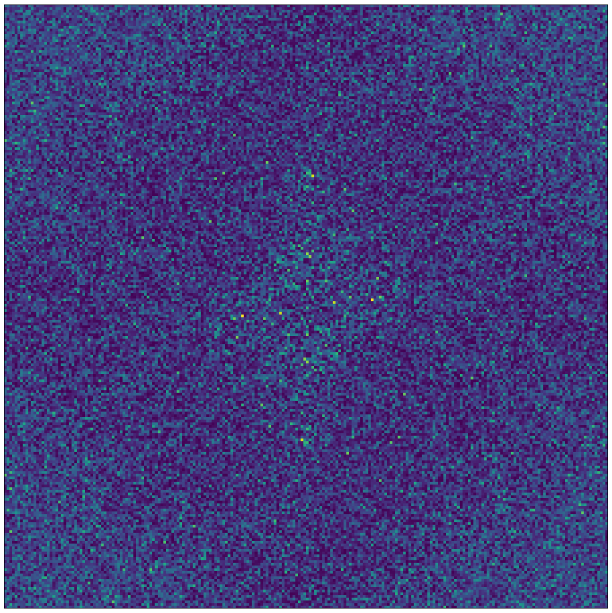
(a) Original



(b) Subtractive diff



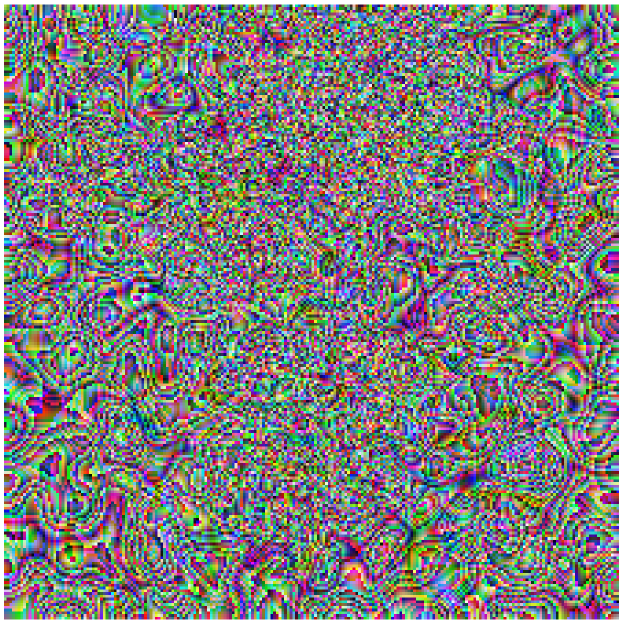
(c) Fourier spectrum



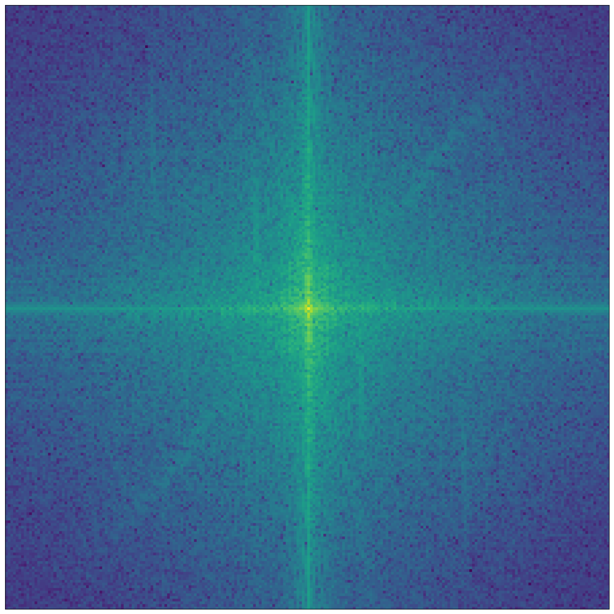
(d) Fourier spectrum Subtractive diff

Figure C.3: Original, Subtractive difference, Fourier spectrum and Subtractive Fourier spectrum for APGD images.

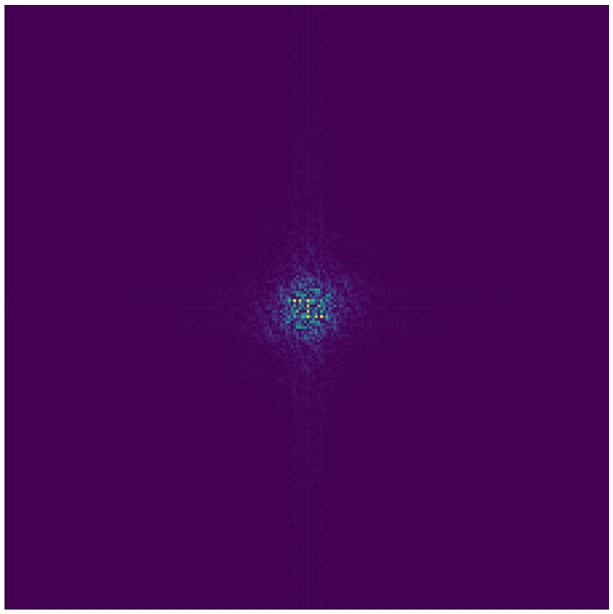
(a) Original



(b) Subtractive diff



(c) Fourier spectrum



(d) Fourier spectrum Subtractive diff

Figure C.4: Original, Subtractive difference, Fourier spectrum and Subtractive Fourier spectrum for shadow images.