Neural Cleanse with Object Detectors

Buyruk, S.A. (Aylin)

Award date:
2021

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computer Science
Security Group

Neural Cleanse with Object Detectors

Master Thesis

S. A. (Aylin) Buyruk

Supervisors:
W. P. A. J. (Wil) Michiels
F. D. (Frits) Schalij
P. L. A. (Peter) Roelse
M. (Mykola) Pechenizkiy

Eindhoven, August 2021

Abstract

Backdoors are attacks that are trained into networks, whereby a small adjustment to the input can cause unwanted behaviour by the network. Due to the complexity and lack of transparency of deep neural networks, backdoors are difficult to detect. One of the methods that has been proposed to identify backdoors is Neural Cleanse, which reverse engineers backdoors in image classifier networks [1]. In this work, we attempted to adapt Neural Cleanse and create a dataset with modified features, such that Neural Cleanse could be used with object detector networks. When this did not function as intended, we approached the problem from another angle. Starting from a working version of Neural Cleanse with classifiers, we recreated our object detector experiments step by step, to investigate where our method was causing problems. We were able to run Neural Cleanse experiments under similar conditions with classifiers as we had with object detectors, showing that the problem did not lie with our method but rather an incompatibility between Neural Cleanse and the complexity of object detectors.

Acknowledgements

I would like to start by thanking my supervisor Wil Michiels for his excellent guidance, and for making my thesis internship an informative and exciting time. Thank you to NXP for granting me this opportunity and all the people I met there for being so welcoming, even when our introductions had to be online. Special shout-out to Frits Schalij for always being ready to help me figure things out with coding, proofreading, advice and ideas. Lastly I would like to thank my close family and friends for their endless support and motivation, and for helping me every time I got stuck because I couldn't think of a word.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Backdoors in deep neural networks (DNNs) are attacks where a small modification to the input causes unexpected or unwanted behaviour [2, 3]. A backdoor can be made by a (relatively small) extension to the training set, after which the trigger is ingrained in the model. Backdoors are difficult to detect due to the lack of transparency in DNNs, and the minimal effect on normal performance of a well trained backdoor. This is a problem because training neural networks is a costly endeavour, both in time and resources. Many organisations outsource this task or use pretrained basic networks which they then specialise according to their needs. If a model is used without knowing its exact origins, or the validity of its training conditions, a legitimate organisation might unknowingly spread a compromised model. Considering the widespread use of DNNs in the modern world, from self-driving cars to network intrusion detection, undetected backdoors could lead to serious consequences.

Different methods have been proposed to detect and/or mitigate backdoors [4, 5]. We focused specifically on a method called Neural Cleanse, which was proposed in 2019 as a generalisable technique to detect and reverse engineer backdoors in DNNs [1]. In the original paper, the authors worked with DNNs for image classification. In this work, we attempt to use the same Neural Cleanse technique but with object detectors. We edited the example code provided by the authors to work with object detectors instead of classifiers, and adapted our dataset that we train Neural Cleanse with to a layout that Neural Cleanse could handle. This required fixing the location of the backdoor pattern, because Neural Cleanse looks for a pattern in a specific location. When this did not yield usable results, we altered the dataset further by randomising the backgrounds. Despite some improvement, these experiments did not produce the desired results either, and the backdoor was not found.

To investigate what problems could be preventing Neural Cleanse from finding the backdoor pattern in our experiments with object detectors, we gradually recreated the same conditions with classifiers. By beginning with the original Neural Cleanse setup as it was proposed in the paper and provided in the sample code, we had a starting point for our tests [6].

We concluded our work by running an experiment with classifiers, with the same setup as we had previously used with object detectors, where Neural Cleanse was able to find our backdoor pattern very accurately. The inability of Neural Cleanse to do the same with the object detector models under the same conditions leads us to believe that the problem lies in the increased complexity of an object detector compared to a classifier, and that Neural Cleanse needs fundamental changes in order to be effective for object detectors.

Chapter 2

Preliminaries

2.1 Classifiers

Artificial neural networks (ANN) can be used for many purposes. In this work we focus specifically on two types of vision problems; classification and object detection. Image classification using a neural network means that the network is essentially applying a function such that given an input image x, it outputs a y from a specific set of discrete values (i.e. classes). The network typically outputs an array of probabilities, one for each possible class. Each value represents the probability that x belongs to that class. The class correlating to the highest probability is then taken as the result.

Neural networks must first be trained to carry out a task, in this case classifying images. The training is done using a training set of images with corresponding labels indicating the correct class of each image, also referred to as true labels. We define this training set as a set T that contains pairs (X_i, Y_i) for $i = 1, 2, ..., z$, where X_i is an image and Y_i its true label, and z is the size of the training set. The network uses this training set T to 'learn' how to classify images by example. A neural network has a set of weights and biases in its neurons (also called parameters), which dictate the function that is applied to the input to generate the output. During training, these weights are constantly adjusted in order to minimise the cost function. A cost function can be defined in many ways, but it is essentially a method of measuring how far the generated output (e.g. classification by the model) is from the desired output (e.g. the true labels). The aim is to minimise the distance between these two, and therefore to minimise the cost function. A cost function can have more than one part, we focus in particular on the classification error loss function, which determines the distance between the classification by the model and the labels in the training set. In our experiments with classifiers, we used cross-entropy as the loss function [7]. We denote the loss function l, with $l(y, y^t) \in \mathbb{R}$, where y is the classification output, and y^t is the target label. In training, the minimisation of the loss is done over a set of samples by the sum of the single elements. In theory, if the output generated by the network exactly matches the desired output, then the loss of the set of samples would be zero. In practice, this situation is very unlikely. For example, the neural network might correctly classify a training image, but for the loss to be zero, the probability output by the network of the correct class would need to be 100% and all other classes 0%[1]. Figure 2.1 can give an idea of how the training process is structured.

The network that results from a training can also be called a model. The performance of a model can be tested using a test set of images and true labels that are separate from the training set. Since these images were not used in the training, they are new to the model and can therefore be used to simulate how it would perform with new data.

For our experiments involving classifiers, we used the GTSRB dataset (see Section 2.3.1). Details about the structure of the classifier models we used can be found in Appendix A.

[1]Note that these conditions would only apply to make the classification loss part of the cost function zero, the cost function may have additional terms for things such as regularisation.

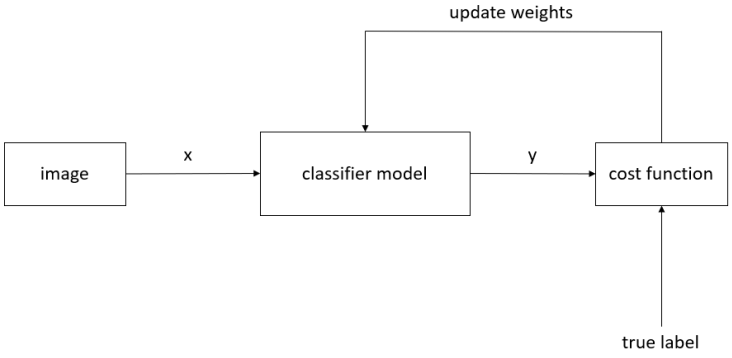


Figure 2.1: Training process of an image classifier model

2.2 Classification vs. Detection

As previously mentioned, we will use two types of neural networks in this work. The first was the classifier as explained in Section 2.1, the second is the object detector.

Image classifiers, when given an input image, will output a list of probabilities for the classification of that image. Object detectors are more complicated, because in addition to classifying objects in a given input image, they also provide the location of each object through use of a bounding box. An object detector can also handle more than one object in an image. Where a classifier is generally only trained to recognise one object per image and output the list of probabilities for that object, a detector will locate objects, and attach a list of classification probabilities to each object.

Due to the differences between classifiers and object detectors, the way their performances are measured is also different. For classifiers, we measure the accuracy, since the only metric we need to take into account is the distance between the generated classification and the true label. For object detectors, we must also consider the correctness of the bounding box. The performance of object detectors is measured using the mean average precision (mAP)[8, 9].

Figure 2.2 shows a representation of the training process.

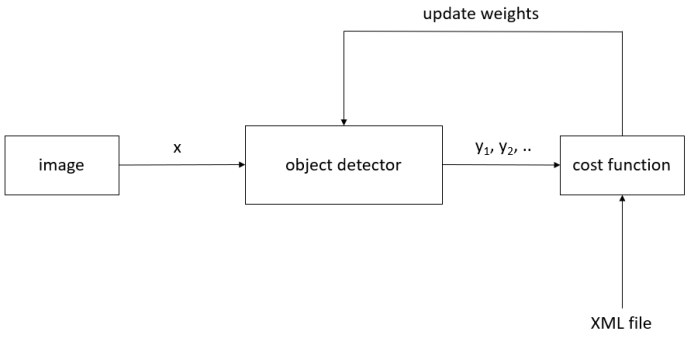


Figure 2.2: Training process of an object detector model

For our experiments involving object detectors, we used EfficientDet [10, 11]. EfficientDet is an object detector developed by a team at Google, optimised with the aim of being efficient and scalable. EfficientDet uses EfficientNet in its architecture (as the backbone), a type of CNN for image classification that was developed by the same team [12]. The implementation of EfficientDet

that we worked with uses focal loss and smooth L1 loss functions as part of its cost function, to measure the classification error and the correctness of the bounding box [13, 14]. The exact definitions of focal loss and smooth L1 loss are left out for readability.

For our experiments involving object detectors, we used the PASCAL VOC dataset (see Section 2.3.2). Details regarding the parameters we used to train our models can be found in Appendix B.

2.3 Datasets

2.3.1 GTSRB

The German Traffic Sign Recognition Benchmark (GTSRB) dataset is a set of 39209 training images and 12630 test images of traffic signs taken in Germany [15]. There are 43 classes of traffic signs, and the images are sorted into directories according to their class. Each image contains only one object.

The number of images per class varies greatly, with the lowest around 200 training images and the highest around 2000. The top 3 classes with the most total images are 13 (yield) with 2970 images, 1 (speed limit 30) with 2940 images, and 2 (speed limit 50) with 2910 images. Most classes can be correctly identified by our classifiers with an accuracy of $> 90\%$, for more details see Appendix C.

The dataset also contains CSV files with annotations such as the coordinates of the bounding box of the object in each image. However, since we only used this dataset in experiments involving classifier models, the annotations were not used.

2.3.2 PASCAL VOC

The Pattern Analysis, Statistical Modelling and Computational Learning (PASCAL) Visual Object Classes (VOC) project was launched in 2005 in the form of yearly challenges to evaluate performance of new methods for creating object detectors [16]. The dataset consists of images and XML files, each image has a related XML file of annotations. These annotations contain information about the image itself such as the size and the source, but also contain information about the objects found in the image such as their class, size, and coordinates of their bounding box. There are 20 object classes: aeroplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, diningtable, dog, horse, motorbike, person, pottedplant, sheep, sofa, train, tvmonitor. Each image can contain more than one object.

The dataset we used in our object detector experiments was originally two separate sets; 2007 and 2012. The 2007 set was split 50 : 50 into training/validation and testing sets, the 2012 set only contained training/validation data since the test image annotations were not made public. We merged the training/validation data of both sets to create one large training set of 16551 images, containing 44169 objects [9, 17]. The test set had 4952 images with 12032 objects. The largest object class by far was the person class, with 17784 objects in total (training/validation + testing). The next largest classes were car with 4468, chair with 3908, and dog with 2514 objects.

A reference for average precision (AP) scores per object class for an EfficientDet model trained on the VOC dataset can be found in Appendix D. This data is from a single run, but we observed that these statistics are representative for all runs. The AP can be seen as an indication of how difficult it is for a model to learn that class. The mAP of a model is an indicator of its overall performance, and is calculated by taking the average of the AP of all classes. The AP of each individual class was an important factor when choosing a source and target class, as explained in Chapter 3.

2.4 Backdoors in Neural Networks

Backdoors are a type of attack that can be carried out on neural networks, whereby the network functions as expected by the user for almost all inputs, except those that meet certain conditions [2, 3]. For these inputs, the behaviour of the network deviates from what is expected. Backdoor attacks can be executed on both classifiers and detectors [2]. The attack is carried out by infecting the training data. Our method was to take images from the training set T (as defined in Section 2.1). We edited these images to include the backdoor pattern, and edited their labelling according to the desired backdoor function, to create an extension set E. We then created a new extended training set T^e, where $T^e = T \cup E$, and used it to train new backdoored models.

An example of a backdoor could be an image classification problem where a neural network is trained to classify traffic signs, with a small white square in the bottom right corner as the backdoor pattern, as seen in Figure 2.3. In this case, our extension E would consist of incorrectly classified images with the backdoor pattern on them, e.g. speed limit 50 signs with a small white square in the bottom right corner, labelled as stop signs. Thus the network will learn to function correctly for almost all inputs, including images of stop signs and speed limits, except if it is given an image of a speed limit 50 sign with a white square on it, which will be classified as a stop sign instead.

It is important to distinguish between backdoors and adversarial examples. Both are patterns that can be pasted on an image to cause unexpected or unwanted behaviour by a network, but the difference lies in the method of attack. Backdoor patterns, as explained, must be trained into models. Therefore, the attacker would require access to the training set. The backdoor pattern can also be chosen by the attacker. On the other hand, adversarial examples are not trained into models, but computed [18, 19]. The attacker must search for a pattern that causes the desired disruption, and cannot choose the shape or size. Adversarial examples can be specific to an image, or universal i.e. working for a whole class.

Backdoors are typically small patterns, such that they do not attract attention from humans. For instance, the previous example of a white square on a traffic sign would be small enough that it could be implemented using stickers. Some backdoors are even designed to be invisible to the human eye [20].

Figure 2.3: A speed limit 50 traffic sign with a backdoor pattern

A targeted backdoor attack is one where the unexpected behaviour caused by the attack is specified, such as in the previous example where the incorrect classification caused by the pattern was specifically defined by a 'source' class and a 'target' class. An untargeted backdoor attack, on the other hand, is one where the attack succeeds as long as the expected behaviour of the network is disrupted for those specific inputs. For comparison with the previous example, an untargeted attack would be if the white square caused a traffic sign to be misclassified as any other sign, without any specifics regarding classes. For the purpose of this work, we will only use targeted backdoor attacks, since this is what the original Neural Cleanse paper works with [1].

A backdoor can also have an all-to-one or one-to-one structure. The difference lies in whether there is only one 'source' class on which the backdoor pattern works, or if simply applying the pattern to any class causes it to be misclassified. The original Neural Cleanse paper discusses all-to-one attacks, and the code follows this structure, however we made a different choice. For this project, we chose to work with a one-to-one arrangement because it would require fewer extra

training images to be added, we would only need to add extra images to one class rather than all classes. This, in turn, would lower the risk of incurring a penalty on the general performance of the model. The more the training set is 'poisoned' with edited images, the higher the chances of interfering with the correct learning process of the model. Lowering the risk of a penalty is important, as one of the goals in preparing backdoored models to test with Neural Cleanse is to minimise the drop in overall performance, while achieving a high accuracy on the backdoor functionality.

Finally, we chose to work with backdoors that cause objects to be misclassified, rather than for example causing objects to become undetectable to the neural network. This was due to Neural Cleanse requiring a target class to work.

2.5 Neural Cleanse

Neural Cleanse is a technique developed to detect backdoors in neural networks, specifically for deep neural networks used in image classification [1]. Neural Cleanse finds backdoors by minimising a cost function, similar to how classifiers are trained as explained in Section 2.1. Contrary to the traditional training of models, where each round of optimisation adjusts the weights inside the network, the training stage of Neural Cleanse changes the input given to the model. In each round of training, a pattern is adjusted, overlaid on the images, and introduced to the model until the desired output is achieved. Given a set of images that the model can correctly classify, Neural Cleanse tries to develop a pattern which, when pasted on these images, will force a misclassification. Consider the example mentioned in Section 2.4, of a model that misclassifies speed limit 50 signs as stop signs if they have a small white square in the bottom right corner. We could run Neural Cleanse with this model and a set of test images of speed limit 50 signs. Starting from a randomised pattern, Neural Cleanse would modify the pattern in each round, to find a version which causes nearly all images to be misclassified as stop signs[2]. It should then produce a black background of the size of the input images, with a small white square in the bottom right corner.

For the sake of clarity, in this work we refer to the final result produced by Neural Cleanse as the pattern. However, internally, the result developed by Neural Cleanse is actually split into two parts. One is called the 'mask', which represents the shape, and consists of a set of pixels with a value in $[0, 1]$. We can write this as $m \in [0, 1]^{n \times n}$, where $n \times n$ is the size of the input. The magnitude of the final result produced by Neural Cleanse is derived from the mask m, and is defined as the L1 norm of the mask. We denote this by $k(m)$. The other part of the Neural Cleanse result is an image called the 'pattern', which represents the colour. We write this as $p \in [0, 1]^{n \times n \times 3}$. The mask and pattern are multiplied together to form the 'fusion'. So we define the fusion f as an image where pixel i, j and colour channel c has value $f_{i,j,c} = m_{i,j} \cdot p_{i,j,c}$. The image x, that is given to the network as input, then has the fusion pasted on it such that each pixel i, j and colour channel c has value $x_{i,j,c} = (1 - m_{i,j}) * x_{i,j,c} + f_{i,j,c}$. When we display results of Neural Cleanse in this work, it is this fusion[3]. However, as mentioned, in the rest of this work we refer to the fusion produced by Neural Cleanse as the pattern, as this is a more intuitive term and improves readability.

During the training phase of Neural Cleanse, adjustments to the pattern are made according to the cost function, in order to minimise its output. The cost function c of Neural Cleanse is as follows:

$$c(y, y^t, m) = l(y, y^t) + \lambda \cdot k(m) \tag{2.1}$$

The first part of the equation is the loss function l as defined above, which measures the error in classification using cross entropy[7]. The closer the classification produced by the model and the intended (mis)classification are, the smaller the loss will be. In keeping with the previous definitions, y is the output produced by the model for an image overlaid with the pattern and y_t is the target label of the attack. The second part of the equation has two elements: $k(m)$ and λ. k

[2]The target as set by the authors of the original paper is to achieve successful misclassification of $> 99\%$ of images.

[3]The displayed fusion results have not been edited unless specified otherwise.

is the function to determine the size of the pattern, as explained earlier in this section. The intent is to minimise k, because although a large pattern covering the whole image may easily force a misclassification, this defeats the purpose of finding the backdoor in the model. λ is the weight for k, which is dynamically adjusted during training to increase or decrease the importance of k in the overall cost function. Thus by minimising the cost function c, Neural Cleanse tries to find a pattern that achieves the desired (mis)classification, with a pattern that is as small as possible.

Originally, Neural Cleanse worked with an all-to-one structure, however as explained in Section 2.4, we chose to work with one-to-one backdoors. We achieved this behaviour by only giving Neural Cleanse images from our designated source class to work with, as suggested in Section 7 of the original Neural Cleanse paper[1]. Neural Cleanse also tries to find separate patterns for all possible target classes, however the user can indicate in the code a specific target class to be prioritised using the parameter `Y_TARGET`. In the given example, we would therefore instruct Neural Cleanse to find a pattern that causes the misclassification of the images as stop signs. In a real life situation, a user would likely not know the source or target class and may try all combinations, but due to time constraints, in this work we focus on the correct combination.

Figure 2.4 shows an outline of the Neural Cleanse training process.

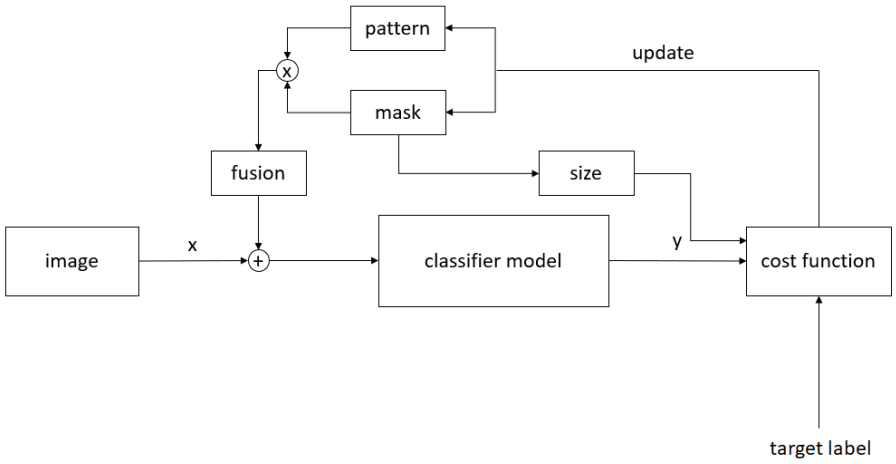


Figure 2.4: Training process outline for Neural Cleanse

The code we use in this work for Neural Cleanse is based on the implementation made available on GitHub by the authors of the original paper [6]. Some changes needed to be made, for example we updated the code from TensorFlow version 1.10.1 to version 2.0 for compatibility with the EfficientDet implementation. Another thing we changed is the intake of input images. The original method of reading all the input images into an array at the start caused memory problems when working with larger images. We changed this to work with a Keras ImageDataGenerator instead, such that the images are only read when they are needed [21].

2.6 Methodology

For each experiment described in this work, we trained three separate models under the defined conditions. The statistics given are the average of the performances of the three. Each Neural Cleanse experiment was also run on all three models. The patterns we show that were generated by Neural Cleanse are from one of the three, but can be taken as representative since there was never a significant difference between the three results.

Chapter 3

Backdoors in Object Detectors

The starting point of the project was to test backdoor functionality in object detectors, and measure a baseline in performance to which future results could be compared. We trained EfficientDet models using the VOC dataset as mentioned in 2.3.2. All the images were resized to 512 x 512 pixels with the OpenCV `resize()` function using bilinear interpolation [22]. To train backdoors into these models, we extended the training set with images and corresponding XML files that were edited according to the desired backdoor pattern and behaviour.

To obtain extra training images that would produce the desired backdoor behaviour, we first filtered the training set to select all images that contain at least one object of the chosen source class. These images were copied and edited to insert the backdoor pattern. An example of how an image is edited can be seen in Figure 3.1. The corresponding XML file of each image was also copied and edited to label each backdoor patterned object as the target class. The edited images and XML files were then added back in to extend the training set. We repeated this process with the test set, but kept these images separate from the original test dataset, such that the performance of the model on the original set and the backdoor function could be measured separately.

(a) (b)

Figure 3.1: A VOC image that has been edited to include the backdoor pattern.

In each edited image, the pattern is in a fixed position relative to the targeted object (i.e. the bounding box) rather than a fixed location in the overall image. This is because images in the dataset can have more than one object present, and therefore the location of the pattern relative to the object bounding box allows us to indicate which of the objects is the target. In addition, we chose to limit the number of backdoor patterns per image to one, to simplify tracking accuracies.

We measure the accuracy of the resulting models using two metrics; accuracy on the problem, and accuracy on the backdoor. The former is measured by the accuracy of the model over the original test dataset, the latter by measuring how many of 'backdoored' objects in the edited test images are classified as the target class.

For consistency with the original Neural Cleanse paper by B. Wang et al. [1], and the example code and model provided with the paper [6], we chose to use a white square in the bottom right

corner as our backdoor. For the size of the square, we chose a 1:4 ratio for both the height and width of the targeted object. For example, an object of 200 x 200 pixels would have a backdoor pattern of 50 x 50 pixels. As explained in Section 2.5, the function of the backdoor needed a source and a target class. The classes 'car' and 'potted plant' were chosen as the source and target class, respectively[1]. It was important to choose a source class with a high AP that the original models without backdoor could identify accurately, because a source class that the models already struggled to recognise would be further negatively affected by the addition of edited training images. This would lead to a worse overall performance of models.

Table D.1 showed that the car class has a high AP. Additionally, as mentioned in Section 2.3.2, cars are present in a large number of images in the dataset, meaning there would be many images to work with. These qualities make car a good candidate for the source class. We found potted plant a suitable target class due to the low similarity between cars and potted plants.

As seen in Figure 3.1, the white square backdoor is placed in the bottom right corner of the bounding box of the car. Figure 3.2 shows an example of the results of a backdoored object detector being given such a test image. The yellow car in front with the white square backdoor pattern on it is now classified as a potted plant instead of a car, however the red and white striped car in the background remains a car.

Figure 3.2: Example detection of a backdoored model

To see how many extra images needed to be added to the training set to properly train the backdoor function into the model, we ran some experiments where the training set was extended with increasing numbers of edited backdoor images. We ran experiments where the training set of 4952 images was extended by 200, 500, and 750 images, where each image contained a car of at least 100 x 100 pixels with the backdoor pattern pasted on it. For pictures with multiple cars, only one backdoor pattern was added, as previously explained. We then tested the resulting models using a test set. This test set was made from every image in the original test dataset that contained at least one car of at least 100 x 100 pixels. This resulted in 558 images, onto which the backdoor pattern was pasted. When testing the models with these images, we recorded for each run how many times a car was detected and how many times a potted plant was detected, using a score threshold of 0.5 and an IoU threshold of 0.5. The score threshold is the minimum confidence that there must be for the output of the model to be counted, so for example if a model is only 40% sure that a detected object is a potted plant, then that won't be counted. The IoU threshold is the minimum score for the intersection over union of the detected bounding box and the true bounding box, it is essentially a measure of how correctly the bounding box is detected by the model. By using these factors, we test that the models detect objects in the correct location, size, class, and with enough confidence in the classification.

[1]For the sake of clarity and readability, we will use conventional names for objects. For example, we will refer to an object of class pottedplant as a potted plant.

Nr of images added	% of cars found	% of potted plants found
200	90.1%	9.0%
500	57.0%	65.2%
750	24.3%	83.3%
834	18.5%	93.2%

Table 3.1: Effect on backdoor performance of increasing the number of backdoored images in training

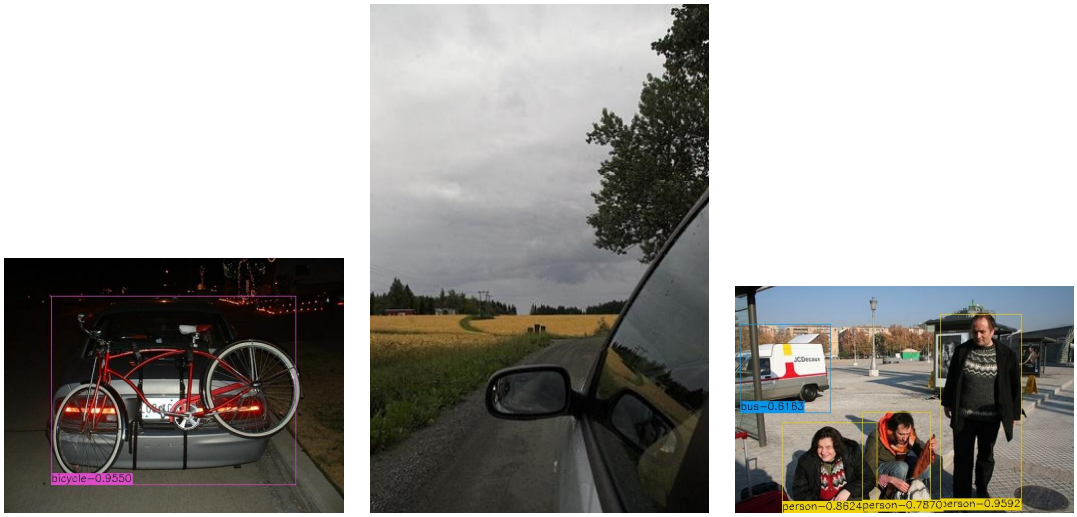
Table 3.1 shows the average results of whether test images of cars with the backdoor pattern were detected as cars or as potted plants. The first row of the table does not add up to 100%, which indicates that in some images the cars with backdoor pattern were not detected as cars or potted plants, or were not detected at all. However the remaining rows add up to more than 100%, which is due to double detections. Figure 3.3 shows examples of situations in which double detections can take place. (a) shows a single object that has been detected by the model as two different classes of objects. The car with the backdoor pattern has been detected both as a car and a potted plant. (b) shows an example where sections of an object have been detected as objects of different classes. In this case the body of the car (with the backdoor pattern included in the bounding box) has been detected as a potted plant, while the open door of the car has been separately detected as a car. All of these situations with double detections can contribute to some rows of Table 3.1 adding up to more than 100%.

(a) (b)

Figure 3.3: Examples of double detection situations

Important to note is that we also ran tests each round with the same image set, but without backdoor pattern, to check if the normal detection rate of cars was affected. In each test the cars detected were compared to the ground truth as recorded in the related XML files. We found that increasing the number of backdoored images added to the training set affected the AP for cars slightly more each time. The performance dropped from an average of 83% (as seen in Appendix D for models that were trained on the original dataset with no additional images, to 71.1% for the models with 750 additional training images. The detection rate of cars in the test images was never 100% as compared to the ground truth data, even for models that were trained on the original dataset. A few examples of factors that influence the detection, or lack thereof, can be seen in Figure 3.4. A car may not be detected because there is another object in front of it (a), or because only part of it is shown in the image (b), or because it is mistaken for a different object instead (c). In the last case, cars that are misclassified are usually labelled as buses instead. The overall performance of the models on the original test set was not significantly affected, as all

models retained an average mAP between $0.70 - 0.73$.



(a) (b) (c)

Figure 3.4: Examples of cars not detected by object detectors

What we can derive from Table 3.1 is that each increase in the number of images causes more potted plants and fewer cars to be detected in the test image set. From this we can conclude that the success rate of the backdoor should improve as more backdoor images are added to the training set.

For the final setup to generate models to be used in subsequent experiments, we extended the training set by selecting every image that had at least one car of at least 100 x 100 pixels. This resulted in 834 extra training images with the backdoor pattern added in. The choice of introducing a minimum size for the cars was motivated by the fact that larger objects are easier for the models to detect. Thus, working with larger objects helps us introduce a backdoor without losing accuracy on the problem, because we hypothesise that better performance on the source class should also improve the backdoor performance. If a model cannot correctly identify an object of the source class, then the backdoor pattern will not cause the desired misclassification.

The final models to be used in the next experiments, with the white square in the right bottom corner as the backdoor, had an average mAP of 0.71. On the backdoored test set, they detected on average 93.2% of cars with the white square as potted plants.

Chapter 4

Neural Cleanse with Object Detectors

4.1 Adapting Neural Cleanse to EfficientDet

To run Neural Cleanse with our object detector, some adaptations needed to be made. As explained in Chapter 2, the in- and outputs of object detectors are different from those of classifiers. Neural Cleanse is not involved with the internal workings of models, but rather it optimises the pattern it generates according to the in- and outputs of the model it is given. Therefore to make Neural Cleanse work with an object detector, the first thing we had to do was adapt its interface with the model, specifically to fit the in- and output structure of EfficientDet. To this end, we edited the code to work with XML files and their contents, such that both the target label(s) and bounding box(es) could be taken in. This information is used in the cost function, which is where we made the second big change. The original Neural Cleanse uses a cost function as shown in Equation 2.1, which takes into account the loss and size of the pattern. There the loss element of the function used to measure the error in classification is cross entropy [1]. However, in our version of Neural Cleanse for EfficientDet, we measure the cost function in three parts. As mentioned in Section 2.2, loss functions used in object detection must not only measure the classification error, but also the correctness of the bounding box. These are the first two parts of the cost function; the third is the size of the pattern. We therefore changed the loss function in Neural Cleanse for object detectors to use the same loss functions as our implementation of EfficientDet. The first is focal loss, which measures the classification error. The second is smooth L1 loss, which essentially measures the error between the bounding box of the object in the XML file, and the bounding box found by the object detector.

By adapting the code in this way, we could ensure that Neural Cleanse tries to find a pattern that does not change the size or location of the bounding box of the located object, only the class. In our version, the target class for each image is obtained from the corresponding XML file. For example, if we would like Neural Cleanse to find a pattern that makes cars be detected as potted plants, we would provide images of cars where the XML files have been edited to label these objects potted plants instead. Neural Cleanse would then use this as the target label in the cost function, and try to find a pattern that makes the output of the model match the XML files.

The general structure of the Neural Cleanse for EfficientDet training process can be seen in Figure 4.1.

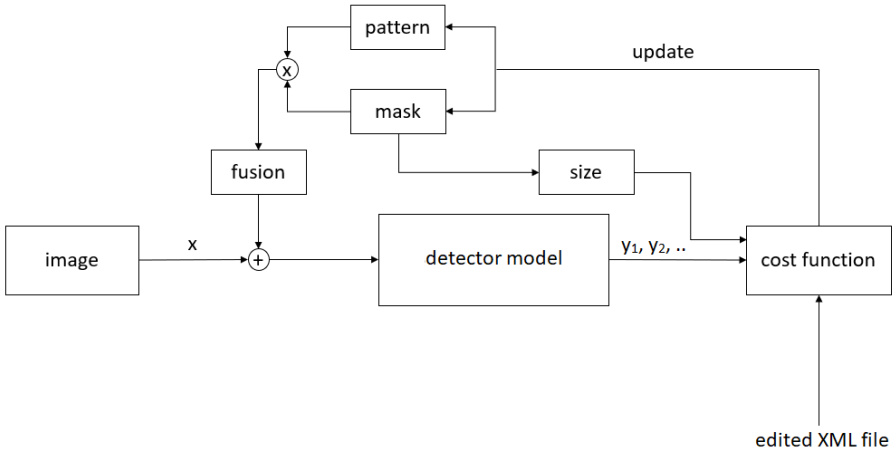


Figure 4.1: Neural Cleanse object detector training process

4.2 Adapting the Dataset

As explained in Section 2.5, Neural Cleanse attempts to find backdoors by generating a pattern that can be overlaid onto input images to induce backdoor functionality. Therefore, Neural Cleanse works under the assumption that the backdoor pattern will always be in the same location on the image. Images used for object detectors, however, can have objects in any location. Considering our approach of placing the backdoor pattern relative to the object bounding box, this means the pattern can be located anywhere in the image.

To allow Neural Cleanse to work with the object detector, we needed to ensure that the backdoor pattern would always be in the same location in the input images provided to Neural Cleanse. We achieved this by creating a new adapted dataset. This dataset was made to train Neural Cleanse, and was made using images from the VOC test set. It consisted of 'car' objects cut out from the test set images, which were then resized to 200x200 pixels and pasted on to a black background of 512x512 pixels. Once again, only cars that are originally at least 100 x 100 pixels were used, resulting in 558 images edited from the test set. An example of this process and the resulting images can be seen in 4.2. In these images, the backdoor pattern which is always 50 x 50 pixels in the bottom right corner of the object, is now also guaranteed to be in the bottom right corner of the overall image. This should allow Neural Cleanse to work with the dataset, because the pattern will always be in the same location.

Before trying to run this new dataset with Neural Cleanse, we first tested it with the detector models to make sure that they could still detect the cars in the images. If the models could no longer detect the cars in this dataset, then Neural Cleanse wouldn't work either. We also tested the functionality of the backdoor by inserting the backdoor pattern on the test images, and checking if the models found potted plants.

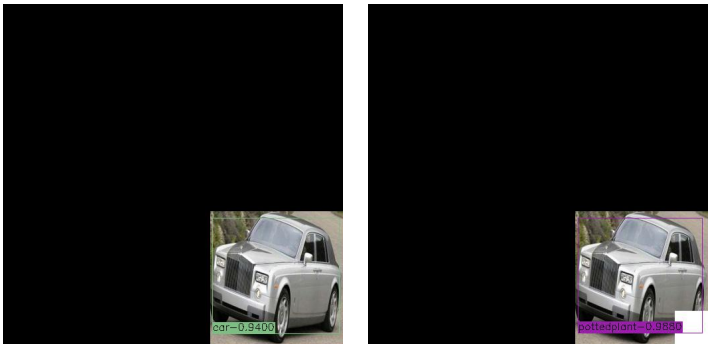
(a) (b)

(c) (d)

Figure 4.2: Example of a car that has been cut out according to it's bounding box and pasted in the bottom right corner of a black background, both without and with backdoor pattern

4.3 Results

The object detector models performed well on the black background dataset. Examples of detections can be seen in Figure 4.3, where (a) shows the detection of the car, and (b) shows the detection of a potted plant when the backdoor pattern is added to the car. In the black background pictures without backdoor pattern, the models found a car in 90.0% of the images, and potted plants in 13%. In the black background pictures with the white square backdoor, the models detected a car in 7.1%, and a potted plant in 95.7% of the images. These results are comparable to those in Chapter 3, meaning that pasting the objects on a black background did not have a significant effect on the performance of the models.



(a) (b)

Figure 4.3: Object detector model detections on the black background dataset, with and without backdoor

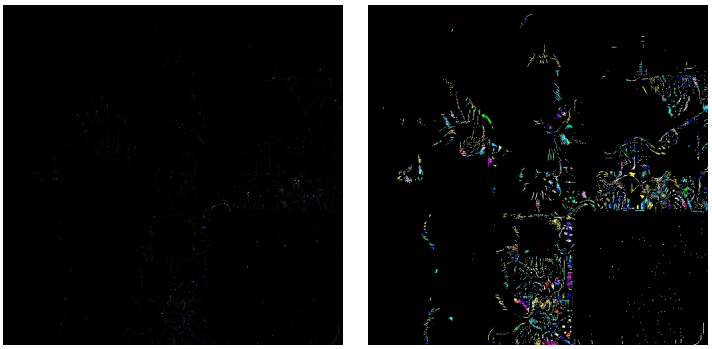
For both experiments, with and without backdoor pattern, the totals add up to more than

100%, which indicates double detections. Upon inspection, the majority of double detections were of the kind where objects are detected as being of two different classes simultaneously. In this case, some cars were being detected as both cars and potted plants in the same image.

After confirming that the object detectors could still work with images where the objects were cut and pasted onto a black background, we used this new dataset for Neural Cleanse. We separated 50 images for testing purposes and ran Neural Cleanse with the remaining 508. The purpose of this was to have some test images that had not been used in the model training or Neural Cleanse process. They could then be used to verify if the pattern produced by Neural Cleanse truly worked on previously unseen images.

Figure 4.4 (a) shows the pattern produced by Neural Cleanse, and Figure 4.4 (b) shows a brightened version of the same result which is easier to see with the human eye. The brightened version was produced by multiplying the brightness of each pixel by 10. We can see immediately that this pattern is not our small square backdoor. Our goal is to have Neural Cleanse find a pattern that is close to the shape of the backdoor we trained into the models, that has the same functionality. In this regard, the experiment has failed because the pattern is the wrong shape. However, Neural Cleanse may have found an adversarial example. The pattern in Figure 4.4 may not look like our backdoor and may not be easily visible to the human eye, but that does not necessarily mean it will have no effect. We tested the pattern by pasting it on the images that were used to train Neural Cleanse, and on the 50 separate test images.

Pasting the pattern produced by Neural Cleanse onto any of the images did not trigger the backdoor function, and the cars were not detected as potted plants. This was the case for the images from both the Neural Cleanse training set and the 50 test images we separated. The rate of detection of cars by the models remained comparable to the results mentioned previously, of cars without a backdoor pattern. The differences were within a 5% range. From this we concluded that Neural Cleanse was not able to find the backdoor with the given models and dataset.



(a) (b)

Figure 4.4: Results of Neural Cleanse on EfficientDet with the black background dataset (a), and the same results brightened (b)

The increased brightness in Figure 4.4 (b) makes it easier to see that many of the pixels are activated, but not visible in (a). Most of the pixels in the pattern are located outside the borders of where the pasted object would be in the bottom right corner. It seems that Neural Cleanse tried to find a pattern around the object rather than on it. This could be caused by the fact that the black background remains the same in every image of the dataset. If the object in the corner changes each time, but the black background is always empty, then the background could be a good place to find a pattern that works as an adversarial example for all images. Neural Cleanse could be trying (and in this case, failing) to find a type of adversarial example using this free space. We tried to reduce this by making a dataset where the background is not solid black.

Chapter 5

Neural Cleanse with Random Backgrounds

As explained in Section 4.3, the results from the previous experiment suggest that the black background gives Neural Cleanse a clear area to look for an adversarial example. We tried to reduce this effect by making sure the background of each image in the dataset is different. That way, there would be a lower probability of Neural Cleanse being able to use the area around the borders of the objects to find a pattern that functions as an adversarial example. We came up with two ways of making each background different; by using randomised bitmaps, and by using a different colour each time.

5.1 Random Bitmap Background

We first attempted to give each image a different background by making a dataset the same way as in Section 4.2, except we edited the images to have a randomised bitmap instead of a black background. Figure 5.1 shows an example of such an image.

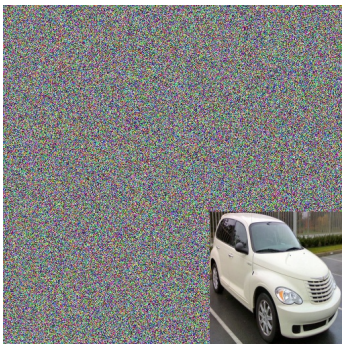


Figure 5.1: Example of a car that has been pasted on a randomised bitmap background

Using the randomised background dataset without backdoor pattern, the models found cars in 89.6% of the images, and potted plants in 17.9%. For the randomised background dataset with backdoor pattern, the models found cars in 9.4% of the images, and potted plants in 94.5%. We see that there is a slight drop in performance of the models on both these datasets when compared to the results of their counterparts with the black background in Section 4.3. However, this difference is in the magnitude of single digit percentage points, and should not affect the performance of Neural Cleanse. The accuracy is good enough that Neural Cleanse should be able to work with this dataset of images with a randomised bitmap background.

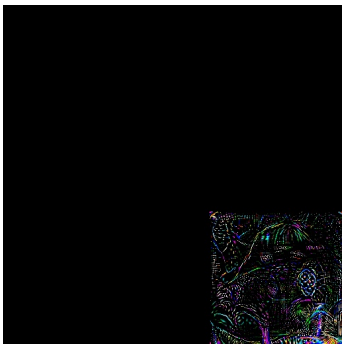
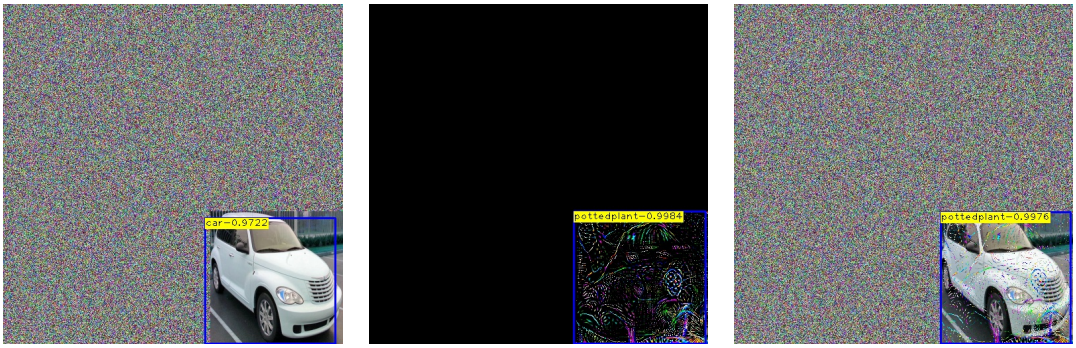


Figure 5.2: Pattern found by Neural Cleanse using the random background dataset

Figure 5.2 shows the result of running Neural Cleanse with the random background dataset. Compared to the result in Section 4.3, there is a clear difference in terms of the pattern being inside the bounds of the pasted objects this time. The randomisation of the backgrounds seems to have worked to force Neural Cleanse to look in the right place. However, we can clearly see that this is not the small square backdoor that we trained into our models.

Looking at the pattern itself, we can see in Figure 5.3 (b) that it is recognised and classified by the models as a potted plant. Overlaying this pattern on a car (Figure 5.3 (a)), also made that car be detected as a potted plant (Figure 5.3 (c)). Using the pattern on the dataset used for Neural Cleanse, 99.8% of the objects were detected as potted plants, and $< 2\%$ cars. For the 50 image test set, the pattern caused 99.3% of the objects to be detected as potted plants, and $< 2\%$ cars.



(a) (b) (c)

Figure 5.3: Effects of the Neural Cleanse backdoor, found with the randomised background dataset

Despite Neural Cleanse finding a pattern that works as a universal adversarial example to mimic the backdoor function, we cannot consider this experiment a success because the original backdoor was not found. As we saw in Figure 5.3 (b), the pattern has visual features that the models recognise as a potted plant. When these features are overlaid on to the cars, they are misclassified as potted plants. However, the original intent of this experiment was to find the square in the bottom right corner, because the purpose of using Neural Cleanse is to find backdoors that have been trained into models. The backdoor we trained into the models was not found, and therefore we consider the experiment not to be a success.

5.2 Random Colour Background

Another method we came up with to make every background different is to give each image a different colour as a background instead of black. We created a new dataset similar to the previous ones, but where each object was pasted onto a randomly chosen solid colour background, as can be seen in the examples in Figure 5.4.

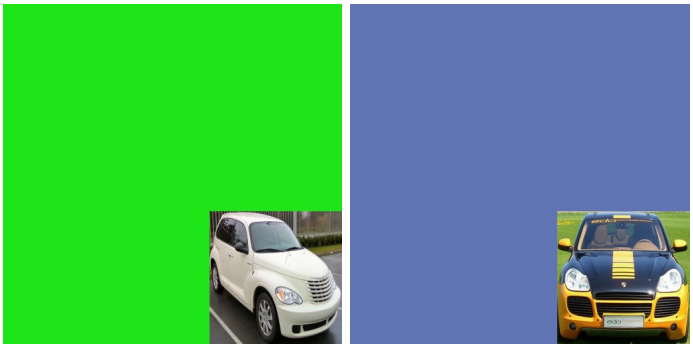


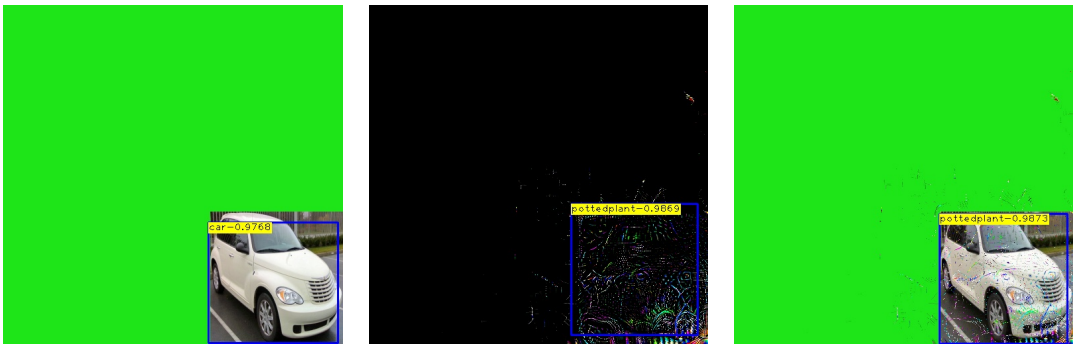
Figure 5.4: Examples of cars that have been pasted on a random colour background

Once again, we tested the performance of the detector models on the new dataset before attempting to use it with Neural Cleanse. For the images without backdoor pattern, the models found cars in 92.8% of images, and potted plants in 19.6%. Using the images with backdoor pattern, the models found cars in 6.2% of images, and potted plants in 96.9%. This performance is comparable to the previous datasets, and good enough that Neural Cleanse should be able to function.

Figure 5.5 shows the result of running Neural Cleanse with the random colour background dataset. This pattern looks similar to the one in the previous experiment shown in Section 5.1. The test results are also comparable. As seen in Figure 5.6 (b), the pattern by itself was detected by the models as a potted plant, meaning once again Neural Cleanse has produced a pattern containing the features recognised as a potted plant. Furthermore, overlaying this pattern on the cars (Figure 5.6(a)) also caused them to be recognised as potted plants by the models (Figure 5.6(c)). After pasting the generated pattern on the Neural Cleanse training set and the separated 50 image test set, the models detected potted plants in 98.1% and 99.3% of images, respectively. There were almost no cars found, at $< 2\%$ for both image sets.

Figure 5.5: Result of running Neural Cleanse using the random colour background dataset

As with the experiment with the randomised bitmap background, we do not consider this result a success. The generated pattern fulfils the requirement of the backdoor function that cars are misclassified as potted plants. However, the original intent of using Neural Cleanse to detect the backdoor in our networks is not achieved.



(a) (b) (c)

Figure 5.6: Effects of the Neural Cleanse backdoor, found with the random colour background dataset

5.3 Potential Problems and Next Steps

The results of our experiments so far showed that Neural Cleanse could not find the backdoor that we trained into the object detector models, despite our attempts to improve conditions. There are a few factors that could potentially be the cause of the lack of expected results from Neural Cleanse with object detectors:

- Layout - Neural Cleanse does not work with the setup where the objects are pasted into one corner of a larger background
- Input size - The total input size of images of 512 x 512 pixels is too large for Neural Cleanse to work effectively
- Object size - The pasted object is too big or too small for Neural Cleanse to be able to work with
- Backdoor size - The size of the backdoor relative to the overall image is too small (or too big) for Neural Cleanse to be able to find
- Complexity of object detectors - Object detectors are more complex than classifiers, and this difference may be the cause for Neural Cleanse not working with object detectors

To determine whether one or more of these listed elements are the cause of the nonperformance, we approached the problem from another angle. So far, the process has been focused on making Neural Cleanse work with object detectors. From Chapter 6 onwards, we arrange a version of Neural Cleanse with a classifier neural network using the GTSRB dataset as presented in the original example code, and step by step change the conditions to match our object detector experiments. By beginning with a working version of Neural Cleanse and testing each possible problem separately, we can determine where the process fails and work on solutions. If we can run the same experiment(s) with classifiers as we did with object detectors, without issue, then perhaps the problem lies with the complexity of object detectors.

Chapter 6

Backdoors in Classifiers

To test the factors potentially causing problems as listed in Section 5.3, we had to begin with a working version of Neural Cleanse that could find the backdoor in a classifier. Then we could move on to test where the process would 'break'. We trained our models using the 32x32 pixel GTSRB dataset and a 4x4 pixel white square backdoor in the bottom right corner as used in the example of the Neural Cleanse paper and code [1, 6]. As explained in Section 2.5, we changed the all-to-one backdoor structure of Neural Cleanse to a one-to-one structure, meaning we had to choose a source and target class for our experiments. As with our choice for the object detector experiments, we took into account the average accuracy per class and the number of images available per class (see Section 2.3.1). We chose 2 (speed limit 50 sign as shown in Figure 6.1) to be our source class and 13 (yield sign as shown in Figure 6.2) to be our target class.

(a) (b)

Figure 6.1: Example image of the source class from the GTSRB dataset (a), and the same image with the backdoor pattern (b).

Figure 6.2: Example image of the target class from the GTSRB dataset

In the original Neural Cleanse paper, the authors state that between $10 - 20\%$ of training images were edited to include the backdoor [1]. However experiments showed that this number was not sufficient to properly train the backdoor function into our models, since we are working with only one source class. We therefore adapted the approach to use 50% of the training images of the source class. In section 2.4, we explained that adding fewer edited images to the training

set was our reason for choosing a one-to-one structure over an all-to-one structure. This reasoning still applies; adding 50% of the training images of one class is still fewer than $10 - 20\%$ of all classes. The extra images were edited to include the backdoor pattern, then used to extend the training set. Images of the source class from the test set were also edited to include the backdoor pattern, but were kept separate so that the backdoor function could be tested independently.

To measure and evaluate the performance of the models trained with this extended dataset, we compared them to the performance of models trained on the original dataset without backdoor. The average accuracy of models without backdoor on the test set was 98.9%, models with the backdoor had an accuracy of 97.3% on the same set. After verifying that the presence of the backdoor did not affect the performance of the models in a significant way, we checked the function of the backdoor. On average, the models classified 89.9% of the backdoored test images as target class 13.

As with the previous experiments, we split the dataset into two parts; the set of images used to train Neural Cleanse, and a separate set of test images. From the total of 750 test images in the source class, we used 700 to train Neural Cleanse and separated 50 to test the pattern performance.

Figure 6.3 shows the result of running Neural Cleanse with this type of model and dataset. The pattern has some noise in the form of extra pixels towards the middle of the image, but we can see the main body lies in the bottom right corner. It may not look like exactly the same pattern to us, however the general shape, size, and location are correct with respect to our backdoor pattern. We trained the models with the white square in the corner as the backdoor, but this does not necessarily mean they learned to recognise the whole square as the backdoor trigger. Certain features or sections may be enough to activate the backdoor behaviour, for example just one corner of the square.

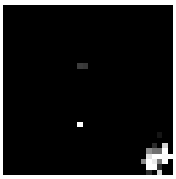


Figure 6.3: Result of running Neural Cleanse with backdoored GTSRB models

The generated pattern performed well when pasted on the test images. When pasted on the Neural Cleanse training set, 99.6% of the images were misclassified as target class 13. Of the 50 test images, 99.3% were misclassified as target class 13. We can conclude that this version of Neural Cleanse works with a classifier and is able to find a pattern similar to our embedded backdoor, which means we can use it as our starting point to test potential problem causing factors.

Chapter 7

Testing the Layout

Of the potential problems listed in Section 5.3, the first one we tested was the layout of the dataset, where we pasted objects in the bottom right corner of a larger background. When working with the VOC dataset in previous experiments, we cut out objects according to their bounding boxes to paste them on to the chosen background. In the case of the GTSRB dataset, there is only one object per image, so the images could just be pasted as they were. We experimented with objects of 32 x 32 pixels pasted on a black background of 64 x 64 pixels, as can be seen in Figure 7.1. The backdoor pattern is 4 x 4 pixels.

(a) (b)

Figure 7.1: Example of an image from the GTSRB dataset that has been pasted on to a black background of 64 x 64 pixels (a), and the same image with the backdoor pattern in the right bottom corner (b)

Important to note is that in previous experiments with EfficientDet, the same models trained with the original training set could be used for subsequent experiments because the object detectors could handle objects moving around, and could still detect objects when the background around them changed. In the experiments with the classifier, to achieve sufficient accuracy and usable results, the models needed to be trained with images with the same layout, the same background, and objects pasted in the same location, as we use for Neural Cleanse. In this case, that means we trained the models using a version of the training set where all the images were also pasted on the black background. The models trained on this dataset performed similarly to the previous backdoored models. They had an average accuracy of 97.2% on the test set, and 92.3% on the backdoored test images.

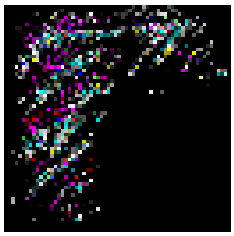


Figure 7.2: Result of running Neural Cleanse with GTSRB images on a 64 x 64 pixel black background

After confirming that the models functioned well, we ran Neural Cleanse. The resulting pattern can be seen in Figure 7.2. This is clearly not the small square backdoor pattern we trained into the models. It seems Neural Cleanse looked for a pattern in the black background surrounding the pasted object, similar to what happened with the object detector experiment with a black background. Contrary to that experiment, however, this generated pattern does work as a type of universal adversarial example that mimics the backdoor function by forcing the misclassification of class 2 objects as class 13. Using this pattern lead to 98.4% of the Neural Cleanse training images being misclassified as class 13, and the same for 96.0% of the 50 test images.

One of the things we still had to test was the working of Neural Cleanse with classifiers trained on images with random backgrounds. We predicted this would cause Neural Cleanse to search for a pattern inside the boundaries of the pasted objects, as was the intent with the corresponding object detector experiments. That would mean this layout of pasting objects on a larger background could still work to find the original backdoor pattern, as long as the background was not just plain black. We tested both alternatives from the object detector experiments from Chapter 5, one with a random bitmap background and one with random colours.

7.1 Random Bitmap Background

The first random background we tested was the randomised bitmap, as used in Section 5.1 where each object was pasted on a (different) randomised bitmap background of 512 x 512 pixels. For this experiment, we used randomised bitmaps of 64 x 64 pixels and pasted objects of 32 x 32 pixels in the bottom right corner. We kept the backdoor at 4 x 4 pixels. These are the same sizes and ratios as in Figure 7.1. Models trained on this randomised dataset had an average accuracy of 97.1% on the test dataset, and 93.1% on the backdoored images.

Running Neural Cleanse with these models and the corresponding dataset lead to the pattern shown in Figure 7.3. There is a small pattern in the bottom right corner which could be similar to our backdoor, but is difficult to see with the human eye. There are also some bright pixels further up to the left. Overall, this pattern looks a lot closer to our backdoor, compared to the previous result where the pattern was all around the pasted object.

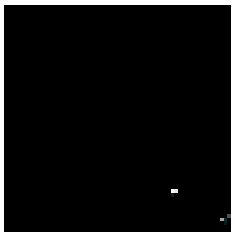


Figure 7.3: Result of running Neural Cleanse with GTSRB images on a 64 x 64 pixel random bitmap background

This pattern performed well with regards to the backdoor function. Pasting this image on the Neural Cleanse training set caused on average 98.9% of the images to be successfully misclassified

as class 13. Of the 50 separated test images, 98.6% were successfully misclassified after pasting the pattern.

7.2 Random Colour Background

We also tested the second random background method as used in Section 5.2, where each object was pasted on a background of a different random colour. We kept the same sizes as used in the previous two experiments, with a background of 64 x 64 pixels, objects of 32 x 32 pixels, and backdoors of 4 x 4 pixels. Models trained on this dataset performed with an average accuracy of 96.1% on the test set and 93.6% on backdoored images.

The result of running Neural Cleanse with these models and this dataset can be seen in Figure 7.4. A pattern with size and shape similar to our backdoor can be seen in the bottom right corner, as well as some other bright pixels a little further up to the left. Once again, this pattern looks like our backdoor and is an improvement over the results of the experiment with the black background, where the pattern was all around the pasted object.

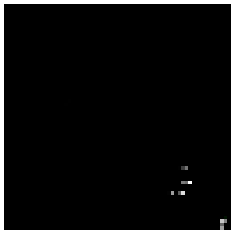


Figure 7.4: Result of running Neural Cleanse with GTSRB images on a 64 x 64 pixel random colour background

The pattern in Figure 7.4 performed well when tested for the backdoor function. It caused 91.4% of the Neural Cleanse training set to be successfully misclassified as class 13, and the same for 98.6% of the 50 separate test images.

Both of the methods of intruducing an element of randomness into the background showed improved results compared to the experiment with the black background. Neural Cleanse succeeded in finding the small square backdoor in the bottom right corner in both cases, and the patterns showed the same function as our backdoor. From this we can deduce that using images with the layout of pasting objects in the corner of a larger background should not cause a problem for Neural Cleanse if a random bitmap or random colour background is used, considering that it was able to find our backdoor in these experiments.

Chapter 8

Testing the Size

The next possible problem we tested was the input size. The original Neural Cleanse experiments were only done with images up to 224 x 224 pixels, but our object detector experiments were with images of 512 x 512 pixels. To test whether the size of our dataset could be causing problems, we made a version of the GTSRB dataset where each object was resized to 200 x 200 pixels, and pasted on a black background of 512 x 512 pixels. The backdoor pattern in the edited images was 50 x 50 pixels. All these measurements match the sizes used in the object detector experiments. Examples of these images can be seen in Figure 8.1.

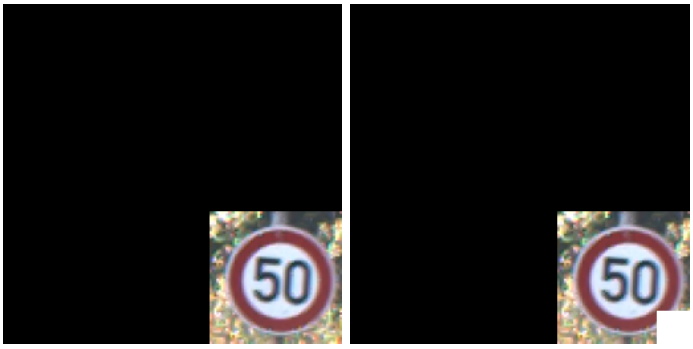


Figure 8.1: Example of an image from the GTSRB dataset that has been pasted in the bottom right corner of a black background of 512 x 512 pixels, and the same image with the backdoor pattern

Unfortunately the size of the images caused the models trained on this dataset to be large, which lead to memory problems with Neural Cleanse. The models had too many trainable parameters, which caused the GPU to run out of memory during training, even with relatively small batch sizes. Initially, we tried to solve this problem by reducing the number of output filters in each layer of the model. However, this lead to a significant drop in accuracy. Instead, we applied a method found in the EfficientDet implementation used in our experiments, where for some layers the stride is set to $(2, 2)$ [11]. By doing the same for the first layer of the classifier models, we reduced the dimensions and thereby reduced the memory usage sufficiently to be able to run Neural Cleanse. The resulting models had an accuracy of 96.1% on the test set and 93.6% on the backdoor images.

The pattern generated by Neural Cleanse using these models and this dataset can be seen in Figure 8.2. The result is not our backdoor pattern, but is interesting to examine. It has a repeating motif which partially resembles our small square. All the brightened pixels of the pattern are clearly outside the area where the objects would be pasted in the bottom right corner, meaning once again Neural Cleanse searched for a universal adversarial example in the black background.

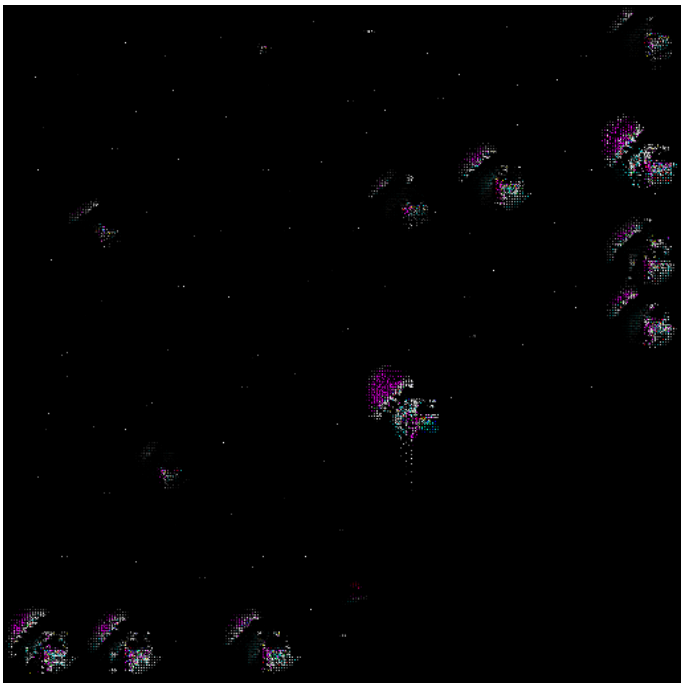


Figure 8.2: Result of running Neural Cleanse with GTSRB images on a 512 x 512 pixel black background

The generated pattern worked well as a universal adversarial example. It caused 99.7% of the Neural Cleanse training images to be misclassified as class 13, and the same for 98.0% of the 50 test images. As with the previous experiment, we predicted that using the random backgrounds for our dataset would cause Neural Cleanse to search for a pattern inside the boundaries of the pasted objects.

8.1 Random Bitmap Background

The randomised bitmap background dataset was made the same way as the corresponding object detector dataset in Sections 5.1, where each object was pasted on a (different) randomised bitmap background of 512 x 512 pixels. The objects were all resized to 200 x 200 pixels, and the backdoor to 50 x 50, for consistency in size and ratio with the previous experiment. An example can be seen in Figure 8.3. The models trained on this dataset performed well, with accuracies of 96.3% on the test dataset and 98.2% on the backdoored images.

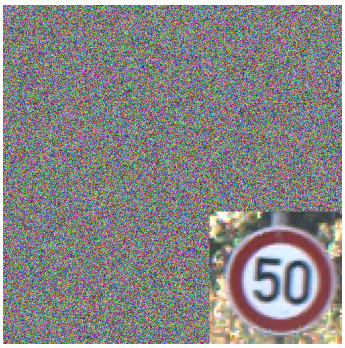


Figure 8.3: Example of an image from the GTSRB dataset that has been pasted on to a randomised bitmap background

Running Neural Cleanse with these models and this dataset lead to interesting results, which can be seen in Figure 8.4. There are some areas with bright pixels towards the middle, but the right border shows some unexpected markings, which were present in all the patterns generated in this experiment. In the bottom right corner we see something that could be part of our backdoor, with a similar size, location, and partial shape. However, it looks like this shape is repeated multiple times along the right side of the image.

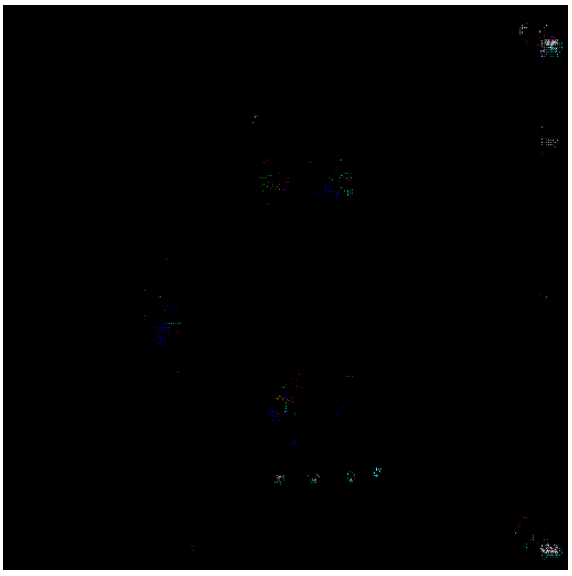


Figure 8.4: Pattern generated by Neural Cleanse for GTSRB images on randomised bitmap backgrounds

The generated pattern did not perform well. Perhaps this is due to the repeating of the pattern in multiple locations, and the extra shapes towards the middle, because the shape in the bottom right corner looks very similar to our original backdoor. Of the Neural Cleanse training set, only 1.8% of the images were misclassified to class 13 as a result of the pattern, and only 2.0% of the 50 test images. This result is interesting because using the same layout with the same background lead to a good result in Section 7.1, where the only difference was the size of the images. The resulting pattern from that experiment with smaller images had the same shape, size, and location as our backdoor pattern, and performed the backdoor function well. In this case, it seems the size of the image makes the problem more complicated for Neural Cleanse, in such a way that it could

not find our backdoor pattern accurately in any of the runs for this dataset with random bitmap backgrounds of 512 x 512 pixels.

8.2 Random Colour Background

We made a dataset where each object was pasted on a different random coloured background of 512 x 512 pixels, as in Section 5.2. Once again, the objects were resized to 200 x 200 pixels with backdoors of 50 x 50, for consistency. Some examples can be seen in Figure 8.5. The models trained on this dataset had an accuracy of 96.4% on the test set, and 95.7% on the backdoored images.

Figure 8.5: Examples of images from the GTSRB dataset that have been pasted on to a randomly coloured background

The Neural Cleanse experiment with this dataset lead to a very clear result, as can be seen in Figure 8.6. The pattern has the same shape, size, and location as our original backdoor pattern, and performed equally well in tests. The generated pattern caused 99.1% of the Neural Cleanse training images to be successfully misclassified as class 13, and the same for 99.3% of the 50 test images. We can consider this last experiment a definite success; Neural Cleanse found a pattern that looks exactly like our original backdoor, and performs equally well.

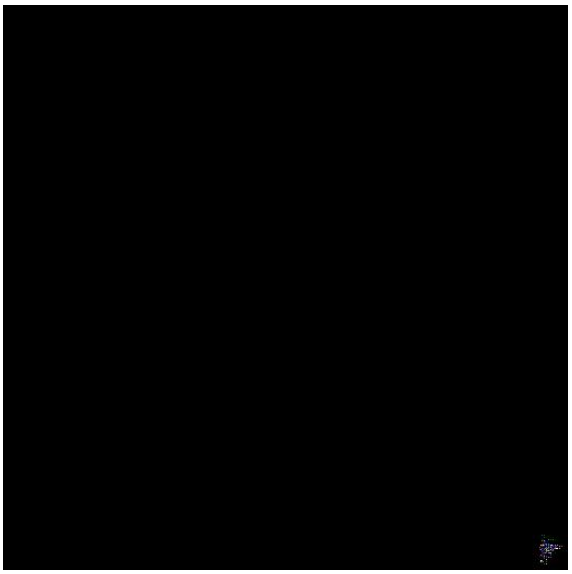


Figure 8.6: Pattern generated by Neural Cleanse for GTSRB images on random coloured backgrounds

Despite the problems with the previous experiment with the random bitmap background, this experiment was performed under similar conditions as the object detector experiment in Section 5.2, and produced similar results to Figure 8.6 for all repetitions. Therefore, we can disregard the input size, object size, backdoor size, and layout as reasons why Neural Cleanse did not work with object detectors.

Chapter 9

Discussion

To explore what could cause lack of results from Neural Cleanse with object detectors, we identified a list of potential problems in Section 5.3, and tested our approach with Neural Cleanse using classifiers, as it was originally designed. We started with a setup where Neural Cleanse worked on classifier models, then we changed the conditions of the experiments one by one to match those done with object detectors. We edited the dataset to have the same layout where objects were pasted in the bottom right corner of a larger background, we increased the size to match the overall size and ratios of the object and backdoor pattern, and we changed the background from black to different random backgrounds. At some stages, Neural Cleanse did not work with the classifier or was only able to find adversarial examples, however we completed the project with the final experiment in Section 8.2, where Neural Cleanse was able to find the exact backdoor trained into the model, which it was not able to do under the same conditions with the object detectors in Section 5.2.

From these results we can conclude that the problem must lie elsewhere. Our hypothesis is that Neural Cleanse needs to be adapted before it can be applied successfully to object detectors.

9.1 Improvements

There is a potential for improvement to our experiments, which is the choice of classes in the detector experiments. In retrospect, potted plant may not have been the strongest choice of target class. It has a low AP, as seen in Table D.1, which means it was a difficult class for models to detect. Adding the edited backdoor images to that class may have served to further 'confuse' the models. This did not really cause problems for our backdoor function, because the car class was fairly strong. Therefore the pattern could still be well trained such that cars with backdoor patterns were classified as potted plants at a high rate. However, it did lead to an increase in cars being detected as potted plants, even without the backdoor pattern (as seen in Chapter 3). This last side effect may have been mitigated with a better target class. In future work, it may be advisable to choose a target class with high AP, where the objects do not resemble cars, such as the cat, dog, or horse classes. The choice of classes is of course only relevant for the analysis of the possibilities and limitations of the Neural Cleanse approach. In practice we would not have this choice, the attacker would decide the source and target class.

Chapter 10

Conclusion

In this work our goal was to use Neural Cleanse with object detectors, by modifying the original code and adapting the dataset. We aimed to adjust to the working of Neural Cleanse by adopting a layout that would fix the location of the object and thereby the location of the backdoor pattern in the images of the dataset. The experiments yielded no usable results at first, prompting us to further edit the dataset by introducing an element of randomness into the backgrounds of the images. Although our adaptations eventually helped Neural Cleanse work and find results that technically satisfied our conditions in the form of adversarial examples, it could not find the backdoor we trained into the models.

We tried to find the problem by approaching it from the other side and recreating our experiments with Neural Cleanse as it was originally developed; with classifier models. We started from a working version of Neural Cleanse that could find the backdoor we trained into classifier models, and then changed the experiments step by step to match our object detector experiments. We tested the layout we used to fix the location of objects, the size of images we used, the size of the objects and backdoor relative to the overall image, and our use of randomised backgrounds. We were able to run Neural Cleanse under similar conditions as our object detector experiments, with a result that matched our backdoor pattern very well. From this we deduced that there must be another cause for the nonperformance of Neural Cleanse with object detectors, one that we did not explore.

We conclude with the hypothesis that Neural Cleanse in its current form cannot be applied successfully object detectors. Our experiments were not sufficient to discover the reason for the incompatibility between Neural Cleanse and object detectors. Future work could continue this investigation by experimenting with classifiers that have been trained with random colour backgrounds as in Section 8.2, but where the object is pasted in a different random location each time, thereby resembling the working of an object detector even more. If the models achieve sufficient accuracy, they could be used for Neural Cleanse. The images used for Neural Cleanse would still need to have the objects pasted in the same location, since Neural Cleanse requires the pattern to always be located at the same position relative to the overall image. Considering all the classifier models in this work were trained with datasets where objects were located in the same place, this future experiment could explore whether the ability to handle objects in different locations has an effect on the working of Neural Cleanse. Another possibility for interesting future work is to try a similar experiment but with different datasets for both the classifiers and object detectors. This experiment could reproduce and confirm the findings of our work. Finally, if the cause of the nonperformance of Neural Cleanse with object detectors could be found and rectified, it would confirm that Neural Cleanse could work with our setup in Chapters 4 and 5 with a modified dataset. The next step would then be to evaluate what changes are needed for Neural Cleanse to be able to work with object detectors with their original dataset.

Bibliography

[1] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 707–723. ii, 1, 5, 6, 7, 8, 12, 20

[2] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, "Badnets: Evaluating Backdooring Attacks on Deep Neural Networks," *IEEE Access*, vol. 7, pp. 47 230–47 244, 2019. 1, 5

[3] Z. Xiang, D. J. Miller, and G. Kesidis, "A benchmark study of backdoor data poisoning defenses for deep neural network classifiers and a novel defense," in *2019 IEEE 29th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, 2019, pp. 1–6. 1, 5

[4] W. Aiken, H. Kim, S. Woo, and J. Ryoo, "Neural network laundering: Removing black-box backdoor watermarks from deep neural networks," *Computers & Security*, vol. 106, p. 102277, 2021. 1

[5] H. Kwon, "Detecting backdoor attacks via class difference in deep neural networks," *IEEE Access*, vol. 8, pp. 191 049–191 056, 2020. 1

[6] B. Wang and S. Shan, "Code implementation of the paper "Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks", at IEEE Security and Privacy 2019." 2019. [Online]. Available: https://github.com/bolunwang/backdoor 1, 7, 8, 20

[7] J. Shore and R. Johnson, "Properties of Cross-Entropy Minimization," *IEEE Transactions on Information Theory*, vol. 27, no. 4, pp. 472–482, 1981. 2, 6

[8] H. Schütze, C. D. Manning, and P. Raghavan, *Introduction to information retrieval*. Cambridge University Press Cambridge, 2008, vol. 39. 3

[9] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes (VOC) Challenge," *Int J Comput Vis*, vol. 88, pp. 303–338, 2010. 3, 4

[10] M. Tan, R. Pang, and Q. V. Le, "EfficientDet: Scalable and Efficient Object Detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 10 781–10 790. 3, 36

[11] Xuannianz, "xuannianz/efficientdet: Efficientdet (scalable and efficient object detection) implementation in keras and tensorflow." [Online]. Available: https://github.com/xuannianz/EfficientDet 3, 25

[12] M. Tan and Q. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," in *International Conference on Machine Learning*. PMLR, 2019, pp. 6105–6114. 3

[13] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 2, pp. 318–327, 2018. 4

[14] R. Girshick, "Fast R-CNN," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448. 4

[15] Institut für Neuroinformatik, "German Traffic Sign Benchmarks," Dec 2010. [Online]. Available: https://benchmark.ini.rub.de/gtsrb_about.html 4

[16] "The PASCAL Visual Object Classes Homepage ." [Online]. Available: http://host.robots.ox.ac.uk/pascal/VOC/ 4

[17] M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge: A Retrospective," *International Journal of Computer Vision*, vol. 111, no. 1, pp. 98–136, 2015. 4

[18] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6572 5

[19] C. Xiao, B. Li, J.-Y. Zhu, W. He, M. Liu, and D. Song, "Generating adversarial examples with adversarial networks," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 2018, pp. 3905–3911. 5

[20] C. Liao, H. Zhong, A. Squicciarini, S. Zhu, and D. Miller, "Backdoor Embedding in Convolutional Neural Network Models Via Invisible Perturbation," *arXiv preprint arXiv:1808.10307*, 2018. 5

[21] Keras, "Keras documentation: Image data preprocessing." [Online]. Available: https://keras.io/api/preprocessing/image/ 7

[22] Open Source Computer Vision, "OpenCV: Geometric Image Transformations." [Online]. Available: https://docs.opencv.org/4.5.3/da/d54/group__imgproc__transform.html 8

Appendix A

Classifier Model

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 30, 30, 32)        9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 15, 15, 64)        18496
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 13, 13, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 6, 6, 128)         73856
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 4, 4, 128)         147584
_________________________________________________________________
global_average_pooling2d (Gl (None, 128)               0
_________________________________________________________________
flatten (Flatten)            (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 512)               66048
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 43)                22059
=================================================================
Total params: 375,115
Trainable params: 375,115
Non-trainable params: 0
_________________________________________________________________
```

Figure A.1: Structure of a classifier model trained on the original GTSRB dataset of 32 x 32 pixels

The structure shown in the figure above is for the classifier models used in our experiments with images of 32 x 32 and 64 x 64 pixels. There is only one difference in the models used for experiments with images of 512 x 512 pixels, the stride of the first layer is set to (2,2).

Appendix B

EfficientDet

Parameter	Value
ϕ	0
batch size	2
epochs	30
steps	10000
learning rate	1e-3

Table B.1: Values of the parameters used for our EfficientDet models.

We used the parameter values given in the table above for all our EfficientDet models. The batch size was kept small to avoid memory problems with the GPU during training. We saved intermediate models at each epoch such that we could select the one with the highest mAP value at the end. In addition, we used an EfficientNet backbone pretrained on the COCO dataset. We froze this backbone during our trainings of the object detector models. For more details about the backbone and structure of EfficientDet models, see the paper by Tan, Pang, and Le[10].

Appendix C

Classifier Accuracy Per Class for GTSRB

Class	Accuracy	Class	Accuracy	Class	Accuracy
0	0.97	15	1.00	30	0.98
1	1.00	16	0.99	31	0.97
2	1.00	17	0.87	32	0.98
3	0.99	18	0.80	33	1.00
4	0.84	19	0.99	34	0.98
5	1.00	20	1.00	35	1.00
6	0.99	21	0.98	36	0.97
7	0.98	22	0.97	37	0.96
8	0.99	23	0.99	38	0.94
9	0.94	24	1.00	39	0.82
10	0.71	25	0.50	40	1.00
11	1.00	26	0.96	41	0.96
12	1.00	27	1.00	42	0.99
13	0.98	28	0.99		
14	1.00	29	0.97		

Table C.1: Average accuracy of 3 classifier models per class of the GTSRB dataset

Appendix D

Object Detector Accuracy Per Class for VOC

Class	AP
cat	0.88
dog	0.87
horse	0.85
car	0.83
train	0.80
bicycle	0.80
motorbike	0.78
person	0.77
bus	0.77
bird	0.75
aeroplane	0.75
cow	0.74
tvmonitor	0.73
sheep	0.70
sofa	0.65
boat	0.61
diningtable	0.56
bottle	0.54
chair	0.46
pottedplant	0.38

Table D.1: Class AP scores for an EfficientDet model trained with the VOC dataset