Evolutionary Methods for Generating Secure Deep Neural Networks

By Daniel Fentham
Student ID: 1251309
For BSc Computer Science

Supervised by Dr. John Bullinaria

April 7, 2019

School of Computer Science
University of Birmingham

Abstract

Adversarial attacks on Deep Neural Networks are a systemic problem in the application of this Artificial Intelligence technology in security critical systems. Previous work has looked at evolutionary methods to generate adversarial data to trick networks into misclassifying data, but very little work has been done in the use of evolutionary methods to produce more secure networks, which this report attempts to remedy. From some initial tests that have been performed, the evolutionary method in this project indicates at a possible defence mechanism that produces Deep Neural Networks that are more robust to adversarial attacks, while keeping a high accuracy on clean images. Furthermore, the application of Multi-Objective Evolutionary Algorithms in the evaluation of proposed network defences is discussed, and in particular the use that this evaluation method could provide to application developers.

Keywords: Deep Neural Networks, Adversarial Attacks, Evolutionary Defence, Multi-Objective Criteria

Software for this project can be found at: https://git-teaching.cs.bham.ac.uk/mod-ug-proj-2018/dxf209

Acknowledgements

To my family and friends, who have unwavering confidence in me and have lost countless hours to the discussion of this project.

To my supervisor Dr. John Bullinaria who has provided invaluable advice regarding the production of this work (even though he constantly added to my to-do list).

To my fellow students, all of whom have been genuinely interested in the work I have done, just as I have been interested in yours.

Contents

1. Introduction

The research conducted in this project relates to adversarial attacks on Deep Neural Networks (DNNs, also referred to as networks or models) and explores the effect that evolution has on the success of these attacks.

An adversarial attack on a DNN is usually in the form of a perturbation of input data to the network, which can be used to cause the network to classify the data incorrectly either to a class of our choice (targeted attack), or a random different class (untargeted attack). These attacks have been demonstrated on a number of different DNNs and can be used wherever the true data generating function is inherently noisy. Furthermore, they can be constructed to be invisible to the human observer, provided the input data has a large enough number of dimensions.

Adversarial attacks on DNNs have grown rapidly in popularity since the seminal paper by Szegedy et al [1] in 2013. Since then a vast quantity of research has been carried out in this area, across many different types of networks trained to do many different tasks. The success of attacks in a majority of these areas has led to great concern from practitioners of Computer Security and Artificial Intelligence, with the application of this technology in security and safety critical areas coming under intense scrutiny.

We can use a toy problem to more intuitively understand how adversarial attacks work at a high level:

Consider a DNN that can determine the price of a house. This network is trained on all the data available about the house, including the important aspects such as the number of rooms, the size of the house, the amount of land and the geographic location. However, it also takes in more information, such as the weather conditions throughout the year. When the network is trained on this data, it may find a strong correlation in the weather conditions, i.e. most houses priced at over £1,000,000 are in locations that exceed 30 degrees Celsius in the summer. An adversarial example in this situation could exploit this fact, and alter the data we pass to the network to say that the temperature for our adversarial house exceeds 30 degrees in the summer. This would cause the network to produce an incorrect output and value the house at a higher price than its true value. However, since when we look at houses we only see the obvious data (like location, number of rooms etc) then this attack is invisible to us.

The above toy problem is unlikely to work in the real world since we, as rational people, can evaluate the situation ourselves. However, it does make it clear that DNNs take non-salient data into account when finding patterns for classification (also known as feature extraction). If we then consider image classification systems which have many dimensions in their classification output space, then we can begin to imagine how easy it can be to alter data to cause the classification to change in any one of these dimensions.

The toy problem is a fairly mundane attack on a DNN. However, if we consider the increasing use of these tools in a variety of situations, we can begin to think of more dangerous effects of adversarial attacks. For instance, several papers have successfully applied these exploits of DNNs on video classification networks [2, 3, 4, 5], facial recognition systems [6] and ad-blocking algorithms [7]. This application of the core research into further areas shows, firstly, how widespread DNNs are being used in real-world applications, and secondly, that the impact of this research has real and worrying consequences on future technology. This is further enforced by a number of examples such as research by Li et al. [2] and Sultani et al. [3] who successfully applied adversarial perturbations to video clips which were passed to classifiers that classified actions occurring in a clip. This would have obvious impacts on CCTV systems that are now being trained to detect malicious behaviour and autonomously report it to authorities. The work by Etimov et al. [5] used printed adversarial stickers placed on stop signs which successfully fooled an image recognition system into believing the stop sign was a speed limit sign. This attack moved the adversarial perurbation to the real world, rather than just resticting it to software. When considering the growing acceptance of autonomous vehicles, the effects of this could be catastrophic. In a similar attack, Sharif et al. [6] applied adversarial perturbations to facial recognition systems where the focus of the perturbation was in the physical environment again, rather than in the data processing. This experiment produced a pair of glasses that could be used for either impersonation (where the classifier misclassified the target as another targeted person) or dodging (where the classifier misclassified the target as another random person). This research has a direct effect on security systems, such as automatic facial recognition systems, being used by law enforcement around the world. These attacks highlight the potential threat to future technologies and the dangers of

quick adoption without considering the possible insecurities of these technologies.

Inspired and shocked by the applications discussed, this report looks at a possible way of reducing the effectiveness of these attacks by considering evolutionary methods of generating classifiers. To my knowledge, there has been no study of the effect of evolutionary methods for generating image classification networks on the susceptibility of DNNs to adversarial attacks. In addition, in the literature surrounding this topic there has been a universal acceptance of a trade-off between *"standard"* accuracy and *"adversarial robustness"* [8]. The standard accuracy of a network refers to the accuracy of a classifier on clean, unperturbed images, whereas adversarial robustness refers to how well a network can ignore adversarial perturbations of images. This trade-off suggests that the robustness of a network is inextricably linked to the standard accuracy, that is a more robust network will invariably have a lower standard accuracy. However, there has been no empirical investigation of this crucial and widely accepted theorem. This is especially important if we consider application developers, who may have to consider trade-offs in the usability of a system, versus the security. We can again examine this using a toy problem:

Consider a company who decides to use a facial recognition system to allow employees access to the building. A system which is secure to adversarial attacks will have a lower standard accuracy, which means employees who should be given access to the building may be rejected more frequently (annoying the legitimate employees), but it is unlikely that any malicious actors will be able to gain access to the building using adversarial methods. On the other hand, the system can be much easier to use for the employees if we increase the standard accuracy. However, this would then decrease the robustness of the system and lead to a higher probability of malicious actors gaining access to the building.

This trade-off is common in security applications, and with the increased use of DNNs in security centred systems, this must be seriously considered by application developers with different approaches taken to the design and training of the DNN depending on the requirements of the system. Many different techniques can be used to secure networks, with each having benefits and drawbacks. Therefore, a method of evaluation of these techniques would be crucial to developers. Since we now have multiple objectives in play, we can frame this as a multi-objective optimisation problem, where one solution cannot be declared to be better than any other.

These ideas lead to a number of project aims:

1) Use evolutionary methods to generate image classification DNNs that have high accuracy and are resistant to adversarial attacks.
2) Explore the possible trade-off between adversarial robustness and standard accuracy.
3) Provide a method to evaluate standard accuracy and adversarial robustness with respect to a network defence.

To explore the aims above a method has been developed to generate, evolve and evaluate image recognition systems which has shown good results in increasing adversarial robustness of networks, whilst also increasing the standard accuracy of the network. Furthermore, methods of using Multi-Objective Evolutionary Algorithms (MOEAs) to evaluate properties of networks (including adversarial defences) with respect to the standard accuracy and adversarial robustness of a network have been developed.

The rest of this report is structured as follows:

- **Research Process** – Details the approach to research and implementation of the project.
- **Previous Work** – An overview of current literature in the areas of adversarial attacks, Multi-objective optimisation and Evolutionary image classification networks.
- **Adversarial Attacks** – Details the mathematics behind adversarial attacks and the intuitive ideas behind them.
- **Implementation** – Details how the project was implemented, from generation and evolution of networks, to the methods of analysis.
- **Results** – Details the findings of this project.

- **Analysis** – Analyses the results and places them in context of current thinking.
- **Further Work** – Details areas in this project that could lead to further work and interesting exploration.
- **Conclusion** – A brief summary of the project as a whole including it's success and deficiencies.
- **References** – The work carried out in this area which was invaluable to this project.
- **Appendix** – Includes the submission structure and the project log.

2. Research Process

Throughout the project I found the management aspect difficult to formalise. At the beginning of the first semester I set out a high level timetable of objectives I wanted to achieve and the dates to achieve them by, but I was cautious about having detailed plans over the work that must be done. Due to the nature of research, I was hesitant to make the plan too rigid since I wanted to have the flexibility to pursue any interesting findings without having huge impacts on any proposed timetables. However, I did keep to the high level deadlines I set myself in the plan and was able to explore interesting findings I discovered in the process of my research.

My approach to this project was as follows:

- I started by reading research papers to assess the current state and trends in the area of adversarial attacks on DNNs.
- From these papers I started to formulate ideas on possible research areas and discussed these with my supervisor.
- I then read deeper into the core topics that would be pivotal in the research and broadly read around related topics to start to draw together research ideas from different areas.
- I was then able to focus on a single research question (guided by my supervisor) and formalise the project.
- I then started some small experiments, mainly putting work I read about into practice (generating adversarial images and classification networks etc).
- The small experiments allowed me to assess the feasibility of my project and loosely determine what would be possible in the time frame.
- The project coding then started, with continued reviews over how best to structure code for easy re-use and ensure longevity of code through the project.

The detail of this process was recorded in logs which can be found in appendix B. The logs also contains comments on vital decisions that were made during the course of this research, and general thoughts on the work that I carried out.

3. Previous Work

3.1. Adversarial examples

The research surrounding adversarial inputs has gained a lot of interest since the seminal paper by Szegedy et al. was published in 2013 [1]. This paper showed a method of computing an adversarial input by maximising the network prediction error. In addition the paper also explained the idea of transferability of these attacks, where adversarial images produced for one network will likely *"fool"* another network (trained to carry out the same task) even if the other network has a different architecture, hyperparameters and even trained on different sets of data. More and more advanced attacks have been proposed with increased effectiveness and subtlety in changes made to the image. A selection of these attacks are discussed later in the paper (see 4) and includes the Fast Gradient Sign Method (FGSM), Basic Iterative Method (BIM), Jacobian Based Saliency Map Attack (JSMA) and Carlini & Wagner L2 attack (C&W L2).

A number of defences to these attacks have been proposed with varying degrees of success against different attacks. One of these is adversarial training proposed by Szegedy et al. [1]. This defence involves training a network on adversarial images with the correct labels attached, which increases the number of training examples used and attempts to avoid the network being fooled by basic adversarial perturbations. Defensive Distillation was introduced by Papernot et al. [9] which expanded on the work of Hinton et al. [10]. This method of defence focuses on making more robust networks, where we begin with an initial network (a teacher) which labels a training set of data with soft labels (the teacher's output vector). We then create a secondary network, identical in size to the initial one (unlike traditional distillation) which is then trained using the soft labels with the training set. The training of the distilled network requires a temperature parameter (*distillation temperature*) which forces the distilled network to be more confident in its predictions.

From the wide reading that has been conducted, the application of evolutionary algorithms is mainly focused on the generation of adversarial images, with very little content regarding the defence of networks. Nguyen et al. [11] used evolutionary methods to generate images which maximised the classification of each class, producing images that classifiers were certain were of a particular class, but were completely unrecognisable to a human. Furthermore, the author has not come across any papers that specifically look at how the evolutionary process can create networks that are more robust to adversarial attacks.

3.2. Trade-offs in accuracy and robustness

One part of this research project is in the possible trade-off between standard accuracy and adversarial robustness which was first presented in a paper by Tsipras et al. [8], which proposes that this trade-off may be inherent. It continues, stating robust networks require more training time and data. The increase in training is due to the augmentation of training data with adversarial examples which has been used in a number of papers [1, 12].

The idea put forward is that adversarially robust DNNs rely on more robust features which align better to human perception and salient data characteristics. The robust networks cannot use features with a correlation below a given threshold, which reduces the number of features used and, therefore, the number of features which can be exploited. The robust networks they discuss have clean feature interpolations which are similar to those found in Generative Adversarial Networks (GANs) which indicates a potential correlation between adversarial robustness and GANs.

In contrast to the Tsipras et al.[8] paper, some networks have been trained on the MNIST dataset, retaining the high standard accuracy, but also being adversarially robust (see Wong et al. [13]). This result could be explained by Schmidt et al. [12] which says that for small classification tasks such as MNIST, there is enough data in the training set to create a strongly generalising network when it is augmented with adversarial images. However, if we consider larger datasets such as CIFAR or ImageNet, then the datasets are not large enough to robustly train a network.

3.3. Multi-Objective Optimisation

A part of this project involves looking at potential ways to analyse defences that can be applied to networks, and the multiple objectives that must be considered in this situation such as standard accuracy, adversarial robustness and (if the selected defence is adversarial training) amount of adversarial training used. Due to the presence of multiple objectives, we cannot say that one solution is categorically better than another, which leads to the generation of Pareto-fronts. Pareto-fronts can be generated by Multi-Objective

Evolutionary Algorithms (MOEAS), which evolve a population of solutions to provide multiple options to the user.

The most well known of these multi-objective evolutionary techniques is the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [14]. NSGA-II is popular due to its ease of use (no user defined hyperparameters are required), time efficiency and wide acceptance in literature as a reliable and accurate evaluation process for multi-objective problems. Specifically, NSGA-II improves upon previous MOEAs with an $O(MN^2)$ time complexity, where M is the number of objectives and N is the population size. It also has an elitism approach which avoids the loss of good solutions if they are found in an early iteration, and no user defined sharing parameter which is present in many MOEAs and controls how well the spread of solutions is maintained. With most problems NSGA-II is able to find a better spread of solutions and better convergence near the true Pareto-Optimal front.

3.4. Evolutionary Image Classification Networks

Neuro-Evolution of Augmenting Topologies (NEAT) [15] has been a popular choice of Neural Evolution algorithms due to its intuitive methods for generating new networks. Work conducted by Real et al. [16] shows how NEAT can be used to produce an image recognition network in one shot which produces a Neural Network which requires no post-processing and with minimal human input. The evolution process uses simplified graphs as the genotype and the mutations are similar to the changes which a human would make to optimise a network architecture such as altering learning rate, inserting convolution layers, altering the stride of a convolution layer as a few examples. The resulting method can produce networks that can classify MNIST and CIFAR images with a competitive accuracy to human constructed networks (94.6% accuracy on CIFAR-10 and 77.0% accuracy on CIFAR-100). This method also has the advantage of a minimal number of parameters that are not optimised by the algorithm.

Neural Architecture Search Network (NASNet) is another method of generating evolved image classification networks introduced by Zoph et al. [17]. This method learns the network architecture directly on the dataset of interest, producing a convolutional cell rather than an entire network, therefore the search for the best convolution architecture is reduced to searching for the best cell structure. These convolutional cells can be found by using small datasets (such as CIFAR) then applied to larger datasets (such as ImageNet) by stacking the cells with different parameters. The NASNet models have been found to be competitive with state-of-the-art human constructed networks, and even exceed their performance in some cases (97.6% on CIFAR-10, 82.7% top-1 and 96.2% top-5 on ImageNet which is 1.2% better in top-1 accuracy than human constructed architectures), whilst carrying out less computations (9×10^9 fewer Floating Point Operations). An important advantage of NASNet is that it generalises feature maps better (a possible property of more robust networks [12]) due to a new regularisation technique *Scheduled Drop Path*, and the general cell structures are more likely to generalise to other applications.

4. Adversarial Attacks

If we consider a Deep Neural Network:

$$\mathbf{F} : \mathbf{X} \mapsto \mathbf{Y}$$

Then adversarial attacks can mathematically be defined as:

$$argmin_{\delta_x} \|\delta_x\| \quad s.t. \quad \mathbf{F}(\mathbf{X} + \delta_x) = \mathbf{Y}^* \tag{1}$$

Where we aim to produce the minimal perturbation (δ_x) to cause a network to change to classification $\mathbf{Y}^*$, which allows the attack to remain invisible to human observers. In this case $\|\delta_x\|$ is the L_2 norm. However, depending on the application we can also consider the L_0 or L_∞ norms, each of which place specific restrictions on the generated adversarial image:

- L_0 – Limits the number of pixels that can be changed, but not the amount that they can be changed by.
- L_∞ – Limits the amount that a single pixel can be changed by, but not the amount of pixels that can be changed.
- L_2 – A balance between the L_0 and L_∞ norms, which limits the number of altered pixels and the amount they can be altered by.

In general the L_p norm can be expressed as:

$$\|v\|_p = \Big(\sum_{i=1}^{n} |v_i|^p \Big)^{1/p}$$

It must also be noted that solving equation 1 is non-trivial since the properties of DNNs make this equation non-linear and non-convex. The adversarial attacks that have been used have been implemented in a widely used Python library used in this field, CleverHans [18]. Since all of the following attacks have direct access to the networks, these are considered white-box attacks. Black-box attacks have shown to be very effective in the production of adversarial images, but they are beyond the scope of this report.

4.1. FGSM

FGSM was introduced by Goodfellow et al. [19] and is an adversarial perturbation that can be calculated with the following simple formula:

$$\mathbf{X}^* = \mathbf{X} + \epsilon \, sign(\nabla_F \, \mathcal{J}(\theta_F, \mathbf{X}, y))$$

In this instance:

$$\delta_x = \epsilon \, sign(\nabla_F \, \mathcal{J}(\theta_F, \mathbf{X}, y))$$

from equation 1 where $\mathbf{X}^*$ is the calculated adversarial input, ϵ is a small constant value, $sign(.)$ is the sign of the value inside the brackets, $\nabla_F \mathcal{J}(.)$ is the gradient of the cost function of then model ($\mathbf{F}$), with θ_F being the parameters of the model, $\mathbf{X}$ is the clean input to the model and y is the expected correct classification value.
FGSM exploits the use of back-propagation to perform gradient ascent away from the classification value y. However, instead of altering the weights (as is usually done in back-propagation) it instead alters the input to correspond to the change in classification output. Due to the function $sign(.)$, only the sign of the calculated gradient is taken (positive or negative) and therefore the movement in the classification space is determined completely by the value of ϵ. A large value of ϵ leads to large changes in the image, which may be further from the true classification value. However, the value of δ_x will be larger, leading to a larger and more noticeable perturbation. Alternatively, a small ϵ value leads to small movements in the classification space, which may lead to the classification being closer to the true classification value, however the perturbation δ_x will be smaller and less noticeable.
This means that in FGSM a balance in the value of ϵ must be struck, to enable the misclassification, whilst keeping the perturbation as undetectable as possible.

4.2. BIM

BIM is an extension to the FGSM method where, instead of taking a single step to generate a perturbation, several small steps are taken, adjusting at each step to lead to more successful misclassifications. A BIM generated adversarial perturbation can be calculated with:

$$\mathbf{X}^*_{n+1} = clip_\epsilon\{\mathbf{X}^*_n + \alpha\ sign(\nabla_F\ \mathcal{J}(\theta_F, \mathbf{X}^*_n, y))\}, \quad \text{where } \mathbf{X}^*_0 = \mathbf{X}$$

Here we have two parameters ϵ and α which indicate the clip value and the step size respectively. After each iteration round, the image is clipped by the ϵ value to ensure it is within the ϵ-neighbourhood of the original image, and therefore ensure the perturbation is minimal.

Due to the iterative nature, smaller steps are taken using BIM and a high successful adversarial rate is still attained. Because of the reduced step size, the image perturbation will be smaller, and therefore less obvious to human observers.

4.3. JSMA

The JSMA attack was first introduced by Papernot et al. [20] with the intention of perturbing only a small number of pixels (they report an average of 4.02% of the input features of a sample must be changed for a successful attack). It also enables heuristic search to be used to generate targeted misclassifications. The algorithm achieves this by changing a single pixel by a variation parameter θ, then checking the effect on the resulting classification by calculating the saliency map by using the gradients of the outputs of the layers of the network. In the generated map, the larger the value, the more effective that change is at fooling the network. To generate an adversarial image, the map is used to change the highly scoring pixels until the target is reached or the maximum number of allowable changes (Υ) have occured.
This approach can be summed up with the steps:

1) Compute the forward derivative $\nabla\mathbf{F}(\mathbf{X}^*)$.
2) Construct a saliency map S based on the forward derivative.
3) Modify an input feature i_{max} by θ (variation parameter).

This process is repeated until the network outputs the desired target vector $\mathbf{Y}^*$ or we reach the maximum distortion value Υ.

We start the process by calculating the forward derivative of the DNN for input $\mathbf{X}$:

$$\nabla\mathbf{F}(\mathbf{X}) = \frac{\partial\mathbf{F}(\mathbf{X})}{\partial\mathbf{X}} = \left[\frac{\partial\mathbf{F}_j(\mathbf{X})}{\partial x_i}\right]_{i\in 1,\ldots,M, j\in 1,\ldots,N}$$

This is the Jacobian function where M is the number of dimensions of the input $\mathbf{X}$ and N is the number of dimensions of the function $\mathbf{F}$. The main difference between this forward derivative and traditional back-propagation is that the derivative is taken of the network directly rather than of the cost function, and differentiation is done with respect to input features rather than network parameters. For a full derivation on how to express $\nabla\mathbf{F}(\mathbf{X}^*)$ in terms of $\mathbf{X}$ and constant values, the reader is referred to the original paper [20].

Adversarial Saliency Maps indicate the most effective input features to change in order to cause the desired misclassification of input. The output of a network has been previously defined as $\mathbf{Y} = \mathbf{F}(\mathbf{X})$ where $\mathbf{Y}$ is the output vector of the network (usually a softmax vector). For the saliency map, given an input $\mathbf{X}$, we get:

$$label(\mathbf{X}) = argmax_{l_t}\mathbf{F}_{l_t}(\mathbf{X})$$

Therefore, a successful adversarial attack means that for an input $\mathbf{X}$ with a true label l_t, then $label(\mathbf{X}^*) \neq l_t$ and for a targeted attack, we must satisfy $label(\mathbf{X}^*) = l_a$ where $l_a \neq l_t$. This can be done by altering input features using the saliency map to increase the probability of l_a and reduce the probabilities of all other classes, so that $l_a = argmax_{l_t}\mathbf{F}_{l_t}(\mathbf{X})$. We use the saliency map $S(\mathbf{X}, l_a)$:

$$S(\mathbf{X}, l_a)[i] = \begin{cases} 0 & \text{if } \frac{\partial\mathbf{F}_{l_a}(\mathbf{X})}{\partial\mathbf{X}_i} < 0 \text{ or } \sum_{l_t\neq l_a}\frac{\partial\mathbf{F}_{l_t}(\mathbf{X})}{\partial\mathbf{X}_i} > 0 \\ \left(\frac{\partial\mathbf{F}_{l_a}(\mathbf{X})}{\partial\mathbf{X}_i}\right)\left|\sum_{l_t\neq l_a}\frac{\partial\mathbf{F}_{l_t}(\mathbf{X})}{\partial\mathbf{X}_i}\right| & \text{otherwise} \end{cases}$$

The value $[i]$ is the input feature under consideration. The first case above sets the saliency map value to 0 in all cases where the change in i causes a decrease in the probability of the target class, or increases the probability of any other non-target classes. The second case sets the saliency value to the product of how much the probability of the target class is increased by and decreases the values of all other classes. Therefore, the final saliency map will indicate how the most effective input features (those with the highest scores) need to change in order to get the desired target label.

This can also be defined in terms of what features an adversary can *decrease* to cause a misclassification, where the only difference is in the second case where the absolute value and constraints of the forward derivative of the target label is taken rather than the other labels.

$$S(\mathbf{X}, l_a)[i] = \begin{cases} 0 & \text{if } \frac{\partial \mathbf{F}_{l_a}(\mathbf{X})}{\partial \mathbf{X}_i} < 0 \text{ or } \sum_{l_t \neq l_a} \frac{\partial \mathbf{F}_{l_t}(\mathbf{X})}{\partial \mathbf{X}_i} > 0 \\ \left|\frac{\partial \mathbf{F}_{l_a}(\mathbf{X})}{\partial \mathbf{X}_i}\right| \left(\sum_{l_t \neq l_a} \frac{\partial \mathbf{F}_{l_t}(\mathbf{X})}{\partial \mathbf{X}_i}\right) & \text{otherwise} \end{cases}$$

We must then select an input i feature using the calculated saliency map and modify it using the variation parameter θ.

Next, we consider the number of iterations we have been through and compare it to the maximum distortion value Υ. If we have reached the maximum number of allowed iterations we terminate, irrespective of whether we have been successful in reaching the target label. High values of Υ means that we are more likely to reach the desired target label, with the disadvantage that the perturbation could be more obvious to the human observer. Alternatively, a low value of Υ leads to a less noticeable perturbation, but a lower chance of reaching the desired target label.

4.4. C&W L2

This method produces adversarial examples with a low distortion in the L_2 metric [21]. To do this, a distance metric $\mathcal{D}(x, x+\delta)$ is used (any one of L_0, L_2 or L_∞). The problem is solved by formulating an optimisation problem. The determination of the objective function f can be seen in the original paper [21]. The focus is to find a solution for w that solves:

$$minimise \quad \|\frac{1}{2}(tanh(w)+1) - x\|_2^2 + c \cdot f(\frac{1}{2}(tanh(w)+1))$$

Where $f(.)$ is the objective function, defined as:

$$f(\mathbf{X}^*) = max(max\{Z(\mathbf{X}^*)_{l_i} : l_i \neq l_t\} - Z(\mathbf{X}^*)_{l_t}, -\kappa)$$

The function above allows us to control the confidence with which the misclassification occurs by changing κ. $Z(.)$ is the output of all layers of the neural network minus the softmax layer (therefore these are the logits). The value of l_t is the target class label and c is determined experimentally using a modified binary search, with the focus of finding c such that $f(\mathbf{X}^*) \leq 0$. The use of $tanh$ ensures that the resulting image will be within valid bounds of 0 and 1, such that $-1 \leq tanh(w_i) \leq 1$, therefore $0 \leq x_i + \delta_i \leq 1$.

5. Implementation

5.1. Network Generation and Evolution

Network evolution focuses on generating networks that are more robust to adversarial attacks by exploring the search space of possible architectures.

Both of the network generation techniques that were detailed in previous work (see 3.4) are powerful evolutionary algorithms that generate complex and sophisticated image classification networks. However, it was decided to use more traditional techniques for this project due to a number of deficiencies they present.

- NASNet – Focuses on evolving convolution cells that can be stacked to generate larger models. However, they are not well defined for smaller networks that will be used in this project, subtle changes to the architecture are also more difficult whilst using NASNet and as such it was decided that NASNet will not be used.
- NEAT – Allows subtle changes to networks as the evolutionary process proceeds, but the process of altering a network is not well supported in TensorFlow or Keras. This would lead to a complex implementation which was not feasible in the timeframe.

Therefore a simpler method of evolution based network generation was used, taking ideas from traditional evolutionary algorithms.

From an initial population of randomly generated networks we start the evolutionary process to reproduce and mutate the networks further. The fitness of the networks is determined by the standard accuracy, with a weighted penalty for the success of adversarial attacks (akin to constraint violations in optimisation problems). This process is summarised in the following sections.

5.2. Network Generation

Networks are generated as a genome, which is a readable string containing the layer information of the network, which is later processed into a Keras model (phenotype). At the end of the evolution process, these genomes are saved in output data (along with other data such as fitness), which allows the Keras models to be created from them at a later date.

Networks are generated by following a template architecture with random aspects included in the creation. A template architecture is used to ensure that the network is feasible and only includes high level rules of the allowed locations of network layers and the correct input and output layers. The process starts by randomly selecting a layer from a set of *'pre-layers'* (layers that must come before a flattening layer), which include layers such as a 2D convolution layer, a max pooling layer and an average pooling layer. A random amount of these layers are chosen (with a maximum number set to 10) before moving on to the *'post-layer'* set. This set contains layers such as fully connected dense layers, dropout layers and average pooling layers.

The full list of layers that can be selected is given below:

- `CV` – Convolution Layer
- `DN` – Dense (fully connected) Layer
- `MP` – Max Pooling Layer
- `AVP` – Average Pooling Layer
- `DO` – Dropout Layer
- `FL` – Flatten Layer

Once a layer is selected we randomly select parameters for the layers. The parameters are restricted in a few cases such as convolution, max pooling and average pooling, since we must ensure the stride chosen still gives a viable image size (and therefore a valid network). Other parameters are randomly chosen in sensible ranges which are explained below:

- Convolution Layer - Number of nodes is restricted to a value between 1 and 128, Kernel size is restricted to values between 1 and 5.

- Dense Layer - Number of nodes is restricted to values between 1 and 128.

These restrictions are applied to try to keep the network sizes small. This ensures they can be trained quickly in the evolution process.

At the end of the process a flattening layer is added to the end of the genome to ensure that the softmax output layer added onto the phenotype is connected to a tensor with the correct number of dimensions. An example of a network genome is:

$$\texttt{AVP}, 7, 3 : \texttt{CV}, 114, 1, 1 : \texttt{CV}, 74, 3, 1 : \texttt{CV}, 95, 3, 1 : \texttt{DN}, 117 : \texttt{DO}, 0.597 : \texttt{DO}, 0.929 : \texttt{FL}$$

This genome generates a network with 95.4% accuracy on the MNIST dataset (the generated model can be seen in figure 1a).

$$\texttt{CV}, 125, 5, 2 : \texttt{MP}, 12, 1 : \texttt{DN}, 10 : \texttt{DN}, 121 : \texttt{DN}, 26 : \texttt{DO}, 0.807 : \texttt{DO}, 0.401 : \texttt{FL}$$

This genome generates a network with 96.6% accuracy on the MNIST dataset (the generated model can be seen in figure 1b).

$$\texttt{CV}, 60, 1, 24 : \texttt{CV}, 85, 1, 1 : \texttt{DN}, 105 : \texttt{FL}$$

This genome generates a network with 11.3% accuracy on the MNIST dataset.

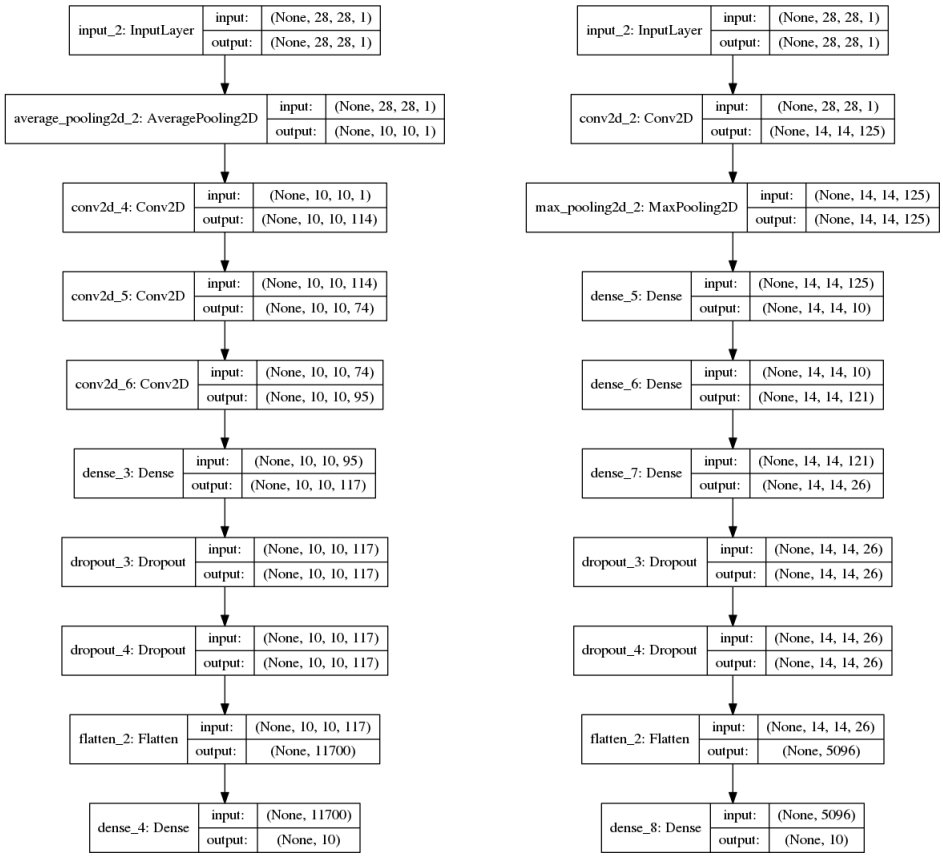


Figure 1: Models of two generated networks

As can be seen from the examples, the network generation can produce networks of varying accuracy on the MNIST dataset. The variation in the accuracy of randomly generated networks is important since we must also consider the robustness of networks to adversarial images. Networks with worse accuracy may

have better adversarial robustness than the high performing networks, therefore it is important to keep less accurate networks to retain the diversity of the population.

5.2.1. Conversion to Phenotype

Once the genotypes have been generated, we can create the phenotypes by taking the encoded layer information and generating a Keras model from them to test. In the creation of the Keras model, we also add an input layer, which allows an input of any size to be specified (28x28x1 for MNIST, 32x32x3 for CIFAR-10) and a softmax output layer with the correct output vector shape for the classifier (1x10 in both MNIST and CIFAR-10). The model can then be trained on the standard training data-sets for MNIST or CIFAR-10 which gives an accuracy 'fitness' value of the network. We can then determine how vulnerable the networks are to adversarial input by generating a set of adversarial images using the networks and then test whether they classify them correctly or not. This result gives an 'adversarial success rate' (an unfitness value) which is multiplied by a weight value α. This is combined with the accuracy value to give a final adjusted fitness value of the network. A full explanation of the network fitness can be found in section 5.3.3.

5.3. Network Evolution

Networks are evolved using crossover, mutation and selection operators applied to an initial population. The evolutionary process is steady state in nature (the population size remains constant) and is non-elitist (the best solutions are not guaranteed to be carried over to the next population). The process itself follows standard evolutionary algorithm implementations quite closely. The methods applied to this problem will be explained in the following sections.

5.3.1. Crossover

Crossover of networks involves the combination of layers between two parent networks to form a child. The parent networks are chosen using a hybrid approach. One parent is selected at random from the set of solutions in the population, with the second being chosen using binary tournament selection. The inclusion of tournament selection guarantees that at least one network will have good performance which speeds up the generation of good solutions.

Two methods for crossover have been implemented, a random pooling process and an annealing pool process.

The random pooling process collects all layers from the genome of the two parents and pools them into a single set of layers. We then randomly sample from this pool, extracting layers and adding them to the new genome with the number of layers selected being a random value between 1 and the size of the set.

The annealing pool follows a similar process except that it contains front and back sets, which keeps the order of the types of layers allowed in the network. The annealing process comes from the selection from each of the front and back sets, where we add a layer to the child model with a linearly decreasing probability. Using this annealing method, we retain a small amount of the network structure in the order of layers, and the network size is determined by random chance (with a limit defined as an annealing temperature parameter). This constructs the child network, an example of which can be seen in figure 2. A child network (figure 2c) is created from the two parent networks (figure 2a, 2b) by the annealing process.

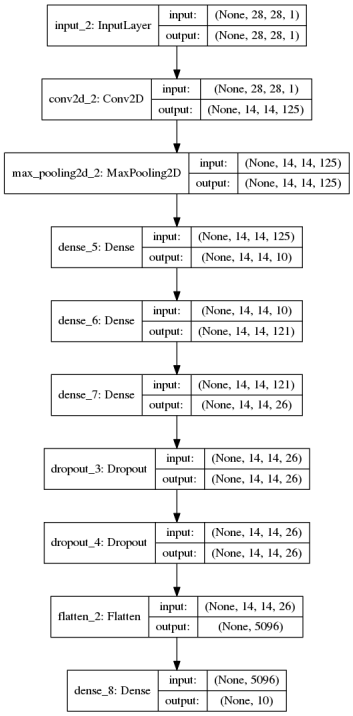
5.3.2. Mutation

Mutation is a simple process which can carry out one of four possible mutations:

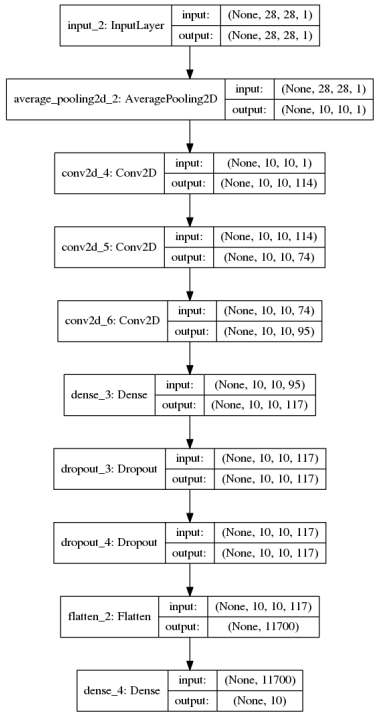
- Alter a random layer
- Add a layer at a random location
- Remove a random layer
- No mutation

Altering a layer takes on different meanings depending on the type of layer selected.

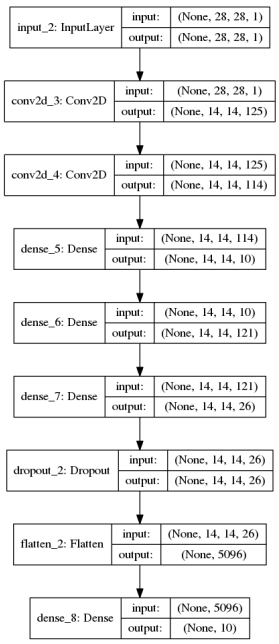
- Convolution layers can have the number of nodes changed, the stride changed and the pool value changed to alter the network slightly.



(a) Parent 1



(b) Parent 2



(c) Child Network

Figure 2: Crossover of two parent models

- Dense layers can have the number of nodes changed.
- Dropout layers can have the dropout probability altered.
- Max Pooling and Average Pooling layers can have the pool and stride values changed.

The insertion and deletion of a layer is fairly simple, since they can just be added or removed from the genome. However, with the addition of layers we must ensure that the resulting strides or pooling of a layer still produces a valid image. To ensure this validity, we take the input size and divide it by each pool size and each convolution stride size to get a "final" image size. The network is considered valid provided the image size is greater than 0 at the end of the division process.

An example of mutations can be seen in figure 4. The initial network (figure 3) is mutated in a number of different ways:

- Figure 4a shows the *altering* of a layer in the network. The effect is not obvious in this representation of networks, but the first convolution layer shows an output shape with the same x and y dimensions, showing that the pool value has been changed from 2 to 1. We can see that this affects the size of the input to the final softmax layer, which now accepts a vector of size 20384 compared to the original 5096.
- Figure 4b shows the *insertion* of a convolution layer immediately after the max pooling layer, which has a pooling of 1 and a stride of 2.
- Figure 4c shows the *removal* of a dropout layer, just before the flattening layer, from the original network.

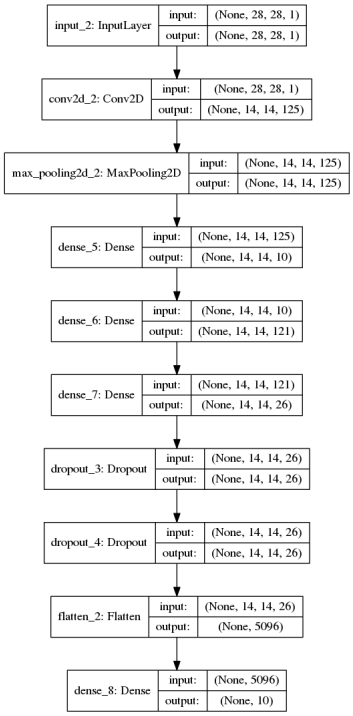
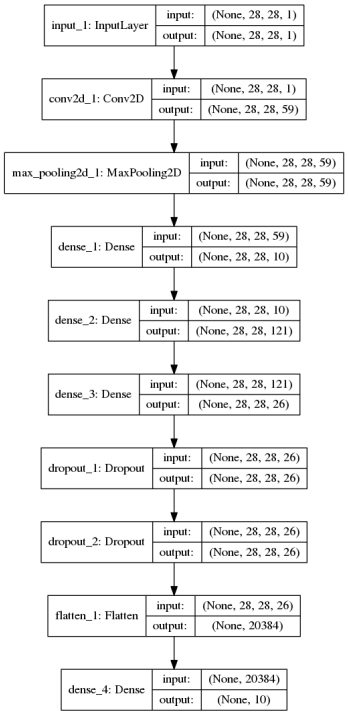
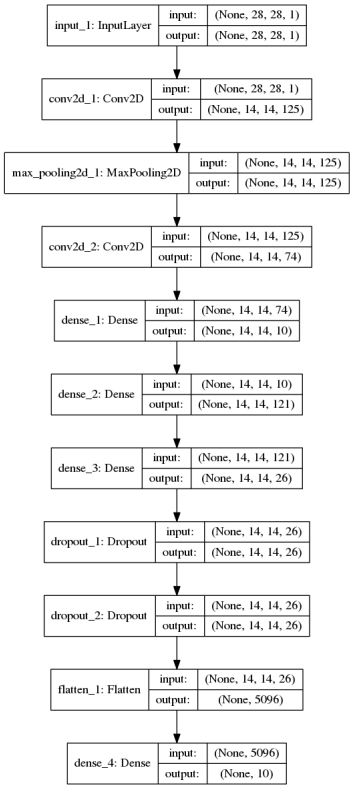


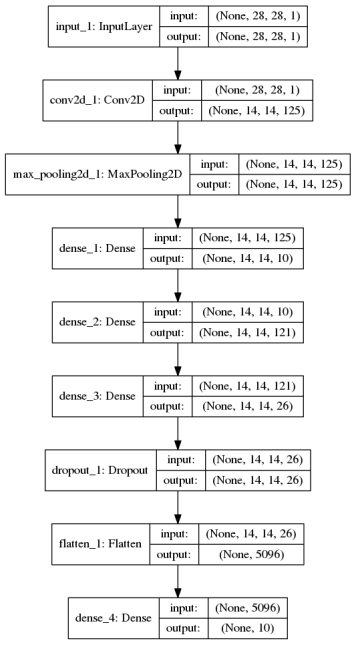
Figure 3: The initial network



(a) Alter Layer



(b) Insert Layer



(c) Remove Layer

Figure 4: Results of various mutation operations

5.3.3. Selection

The selection of networks to produce the next population is done by binary tournament selection. This was chosen since it is the simplest to implement, it is a widely used selection process, and it is not sensitive to the fitness function. We can also alter the selection easily by changing the tournament size which adjusts the selection pressure.

The fitness of a solution is defined as:

$$f'(x) = f(x) - \alpha\Omega(x)$$

Where $f(x)$ is the standard accuracy of the network, $\Omega(x)$ is the adversarial success rate (the percentage of adversarial images a networks classifies incorrectly from the clean class) of attacks on the network (referred to as *unfitness*) and α is a chosen constant value. As a note, usually only a single attack is considered on the network at a time (i.e. we only attempt to defend against a single attack during the evolution process). The selection of α gives us a parameter we can tune, and therefore we can consider how α changes the way networks are selected.

α	**Effect**
$\alpha = 1$	The solution fitness and unfitness are considered equal in importance. The maximum f' of a solution is 1, with minimum -1.
$\alpha > 1$	The unfitness of a solution is considered more important than the fitness. With maximum unfitness (α), the f' of a solution can easily become negative and biases the selection to avoid solutions that have a high unfitness value at the cost of fitness.
$\alpha < 1$	The fitness of a solution is considered to be more important than unfitness. With maximum fitness (1), the f' of a solution will always be positive. This biases the selection to take solutions with higher fitness values, at the cost of allowing the unfitness to be higher.

Table 1: Effect of α on solution selection

Since both the fitness and unfitness measures are on the same scale, there is no need to provide any scaling functions.

The selection process has been chosen to be non-elitist, since elitism introduces selection pressure to the population. This could lead to a few very fit solutions dominating the population as they would have many chances to undergo crossover. This would increase the speed at which we converge to good solutions, but also increase the risk of converging to a local optimum solution, since we would have a less diverse population.

5.3.4. Implementation of evolution in code

To implement the evolutionary algorithm in code we can start with pseudocode of the evolution process:

```
P := generate_initial_population()
evaluate(P)

while(!term) {
    offspring := []
    for(i := 0; i < len(P); i++) {
        p1, p2 := select_parents(P) //Select two parents from the population
        child := crossover(p1, p2) // Generate child from crossover of both parents
        child := mutate(child) //Mutate the child network
        evaluate(child)
        offspring := offspring ∪ child
    }
    P := P ∪ offspring
    P := tournament_selection(P) //Select a new population using binary tournament selection
    term = check_terminate() //Check if a termination condition has been satisfied
}
return P
```

Listing 1: Evolution Pseudocode

When coded, this worked well for small tests of a few networks, but it was incredibly slow. To speed up the process and take advantage of the extra GPU power, multi-processing was used. The best candidate for parallelisation of code is the generation of child networks. This is an independent process that requires access to the global network population to select parents and collect the child networks. This move requires some restructuring of code to ensure we do not encounter race conditions or corrupt data we have concurrent access to. Since no limits were placed on the parent selection (i.e. only a single network can be a parent at most once) we can provide each process a copy of the global network population to select the parent from. This avoids race conditions in the selection of parents and makes the process much simpler. To ensure all child networks are added to the global population properly, all processes wait until all other processes finish, and then each add the generated child to the global network population. In addition to avoiding the possible destruction of data, it ensures that all children are generated from networks from the previous generation, and no network is generated from a child of the same generation. This can be summarised in the following pseudocode:

```
1  func generate_child(P) {
2      p1, p2 := select_parents(P) //Select two parents from the population
3      child := crossover(p1, p2) // Generate child from crossover of both parents
4      child := mutate(child) //Mutate the child network
5      evaluate(child)
6      return child
7  }
8
9  P := generate_initial_population()
10 evaluate(P)
11
12 while(!term) {
13     offspring := [⋃_{i=1}^{i=|P|} generate_child(P)] //Multiple generate_child processes
14     //This section is blocked until all processes have finish generating the offspring list
15     P := P ∪ offspring
16     P := tournament_selection(P) //Select a new population using binary tournament selection
17     term = check_terminate() //Check if a termination condition has been satisfied
18 }
19 return P
```

Listing 2: Introduction of Multi-processing

In practice, line 13 of the above listing is implemented using a map function across a list of integers to generate the child networks in place in the list. This ensures that there is no loss of data as processes try to write to the list, and each process is provided with a copy of the global network population.

5.4. Result Analysis

If we reflect back on the aims that were set out in the introduction (see section 1):

1) Use evolutionary methods to generate image classification DNNs that have high accuracy and are resistant to adversarial attacks.
2) Explore the possible trade-off between adversarial robustness and standard accuracy.
3) Provide a method to evaluate standard accuracy and adversarial robustness with respect to a network defence.

How these points will be tackled has been considered in the sections below.

5.4.1. Evaluation of Network Evolution

The aim of the evolution process is to generate networks which have a high standard accuracy, but also high robustness to adversarial attacks. To test the evolutionary process a number of tests have been run. To start with, the MNIST dataset is used to carry out separate evolution processes to defend against the FGSM attack, JSMA attack and C&W L2 attacks. This has been repeated using the CIFAR-10 dataset to evaluate the effectiveness of the evolution process on a more complex dataset. For each of these tests, graphs have been produced which should indicate the effectiveness of the evolution process, with the ideal graph showing a general increase in standard accuracy (up to a limit) and a general decrease in the success of adversarial attacks. We can also evaluate the effectiveness of the evolution method by comparing results obtained from a random generation of a new population at each epoch to the evolution process at each epoch.

5.4.2. Evaluation of Trade-off

The evaluation of the proposed trade-off will be done passively in the tests carried out. The trade-off postulates that, as the adversarial robustness increases, the standard accuracy of a network decreases. This will be seen clearly in the results collected concerning the network evolution, since if a trade-off does exist, then as the adversarial success rate decreased, we should also see a slight decrease in the standard accuracy. This should become more obvious as more epochs are used.

We should also be able to see this effect when evaluating network defences, since as the defence (potentially) increases the adversarial robustness of a network, the standard accuracy should decrease. This should be visible in graphs produced looking at the effect of the defence on both the standard accuracy and adversarial robustness.

5.4.3. Evaluation of Network Defence

The evaluation of defences with respect to multiple objectives is common in many security related sectors. This naturally leads to an empirical analysis of a network defence with respect to multiple competing objectives.

The results from the experiments performed, will take a number of network properties into account, such as standard classification accuracy, adversarial robustness and a defence mechanism, such as amount of adversarial training. These competing properties of the network create a multi-objective optimisation problem with a number of Pareto-optimal solutions. Since we cannot say whether one Pareto-optimal solution is better than another, we must consider all possible optimal solutions and generate a Pareto-front. Pareto-fronts can be generated using a MOEAs, in this instance NSGA-II is used.

We can summarise the NSGA-II algorithm with pseudocode:

```
Input: P_t, Q_t      //P_t is the parent population, Q_t is the offspring population
R_t := P_t ∪ Q_t
fast_nondominated_sort(R_t)          //Sorts the solutions into non-dominated fronts
i := 1, P_{t+1} := ∅
while (|P_{t+1}| + |F_i| < N) { //Where F_i is the non-dominated front, N is population size
    P_{t+1} := P_{t+1} ∪ F_i
    i++
}
excess := N - |P_{t+1}|    //Get the remaining space to fill
P_{t+1} := P_{t+1} ∪ crowding_sort(F_i, excess) //Include the most widely spread solutions in F_i
Q_{t+1} := crowded_tournament(P_{t+1}) //Generate the next population of offspring
```

Listing 3: NSGA-II Algorithm

The algorithm in the listing above, combines the parent and offspring populations, and adds the first i non-dominated fronts to the next population (P_{t+1}). When we can no longer add fronts to the population (the size of the front is too large to fill the remaining space in the population) we sort the excess values to ensure a good spread of solutions from those which remain, and add these diverse solutions to the next population.

The algorithm above includes some interesting sorting mechanisms such as the fast non-dominated sort which is summarised as:

```
Input: P  //A population P
for each p ∈ P {
    n_p := 0, S_p := ∅
    for each q ∈ P where q ≠ p {
        if(p ≺ q)      //p dominates q
            S_p := S_p ∪ {q}  //Add q to set of solutions p dominates
        else if (q ≺ p)
            n_p := n_p + 1  //Increment the number of solutions that dominate p
    }
    if(n_p == 0)
        p_rank := 1, F_1 := F_1 ∪ {p}
}
i := 1      //Initialise the front counter
while(F_i ≠ ∅) {
    Q := ∅
    for each p ∈ F_i {
        for each q ∈ S_p {
            n_q := n_q - 1
            if(n_q == 0)  // q belongs to next front (no longer dominated)
                q_rank := i + 1, Q := Q ∪ {q}
        }
    }
    i := i + 1
    F_i := Q
}
```

Listing 4: fast_nondominated_sort Algorithm

The pseudocode above sorts the population into fronts by calculating the number of solutions that dominate another solution. Solutions that are dominated by fewer other solutions are added to earlier fronts. This ensures that less dominated solutions remain in the population longer.

The main NSGA-II algorithm along with fast non-dominated sort, crowding sort and crowded tournament selection (which enforces diversity in the population with solutions spread evenly along the Pareto-front) we get more diverse solutions close to the Pareto-optimal front.

Due to time restrictions and the presence of efficient available code, NSGA-II is provided by the the Platypus library[1], which implements a number of MOEAs including NSGA-II.

[1]Library available at https://platypus.readthedocs.io/en/latest/

6. Results

6.1. Preliminary Work

Before collecting results from the evolutionary algorithm that has been designed, appropriate parameter values for training networks and the various adversarial attacks that would be applied to them must be found. These parameters are incredibly important to the success of any machine learning based research, therefore the process has been documented below.

To test the parameters, two networks were used, one trained on the MNIST and the other CIFAR-10. The architecture of these networks was taken from the original work on defensive distillation [9], and also used in the paper by Carlini et al. [21]. The architectures have been changed slightly, since when training, overfitting was observed. Therefore, two dropout layers have been added to each network. In addition, the Adam optimisation technique was used instead of the original stochastic gradient descent, since the networks achieved better accuracy with Adam. The networks can be seen in figure 5 with the standard accuracy of the networks also shown (trained after the determination of learning rate in section 6.1.1). The standard accuracy of both the MNIST and CIFAR-10 networks were close to those achieved by Carlini et al. with their networks reaching an MNIST accuracy of 99.5% and CIFAR-10 accuracy of 80%.

MNIST Architecture
Conv2D+RELU – 32 (3x3)
Conv2D+RELU – 32 (3x3)
MaxPooling – (2x2)
Conv2D+RELU – 64 (3x3)
Conv2D+RELU – 64 (3x3)
MaxPooling2D – (2x2)
Dense – 200
Dropout – 0.8
Dense – 200
Dropout – 0.5
Flatten
Dense – 10 (softmax)

(a)

CIFAR-10 Architecture
Conv2D+RELU – 64 (3x3)
Conv2D+RELU – 64 (3x3)
MaxPooling – (2x2)
Conv2D+RELU – 128 (3x3)
Conv2D+RELU – 128 (3x3)
MaxPooling2D – (2x2)
Dense – 256
Dropout – 0.8
Dense – 256
Dropout – 0.5
Flatten
Dense – 10 (softmax)

(b)

Figure 5: Two test networks

	MNIST	CIFAR-10
Standard Accuracy (%)	99.31	81.27

6.1.1. Determination of Learning Rate

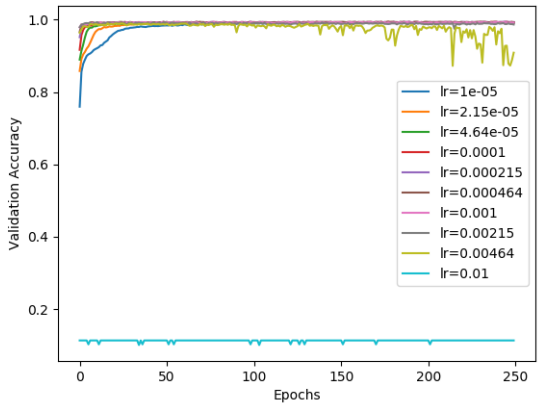
To ensure the generated models were fairly evaluated, a number of experiments were run to determine the optimal learning rates and sufficient number of epochs which should be used to train the networks. The networks will be trained using the Adam optimiser since it has been shown to produce well performing networks on both MNIST and CIFAR-10 datasets [22]. Due to the vast range of network architectures that would be generated by the evolutionary process, fairly simple network architectures were used which were described in the previous section and are similar to the networks that would be generated by evolution. We start by determining the learning rate to use and from some initial observations, learning rates in the range 1e-5 to 1e-2 resulted in networks achieving the highest validation accuracy on the test data. Using these values as a bound, we can generate a list of 10 values between these numbers, equally spaced on a logarithmic scale. The initial observations also indicated a convergence of the model at approximately 100 epochs at a batch size of 32. In this experiment the speed of training was increased by using a batch size of 256. Therefore, to allow ample time for the model to converge, we run the training for 250 epochs while testing these learning rate parameters.

The results can be seen in figures 6 and 7. Figure 6a shows the learning rates that were sampled, the accuracy of the models on the validation set of MNIST images and the number of epochs. As the graph shows, for the highest performing learning rates, the validation accuracy converges very quickly with small

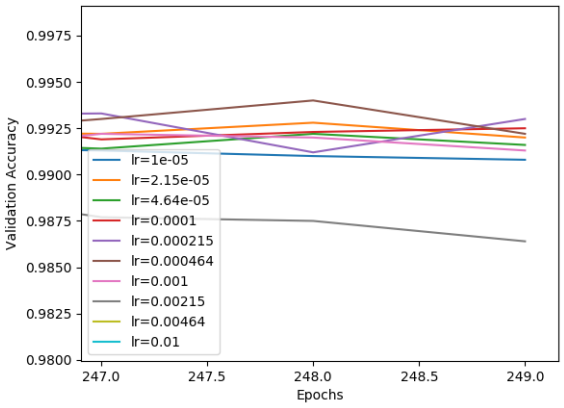
amounts of noise. Figure 6b shows the best performing learning rate is approximately 2e-4, with learning rates in the range of 1e-5 to 1e-3 also producing models with high accuracy. We can also see the effect of learning rate on the time taken to converge in figure 6c which shows that the smaller learning rates take longer to converge (which is expected).

The graphs in figure 7 show the effect of learning rates on the CIFAR-10 model. The figure 7a shows that the optimal learning rates are within the range 5e-5 to 5e-4 with other learning rates falling below approximately 70% validation accuracy. The number of epochs required for convergence appears to be between 150 to 200 epochs depending on the learning rate chosen. The logarithmic view in figure 7b shows the slower convergence of smaller learning rates again.

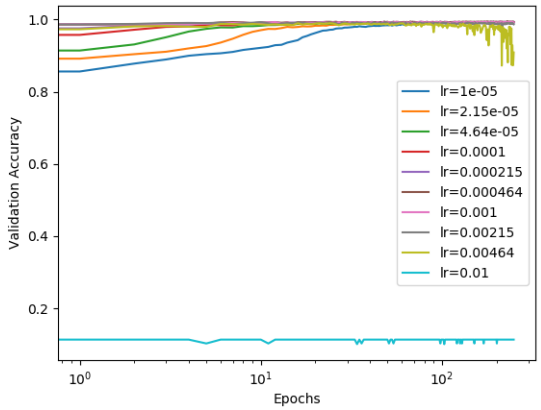
This test has shown the optimal learning rates and number of epochs for the different types of networks that will be trained. From the results, when training MNIST models, a learning rate of 2e-4 will be used, for between 100 to 150 epochs (to ensure convergence). When training CIFAR-10 models, a learning rate between 5e-5 to 5e-4 will be used, along with between 150 and 200 epochs.



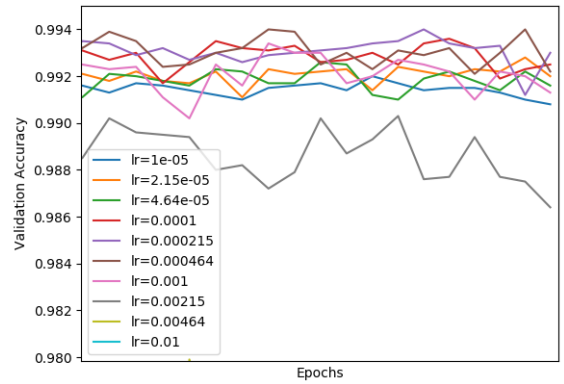
(a) The result of altering learning rate on MNIST model



(b) A closer view of the top performing learning rates

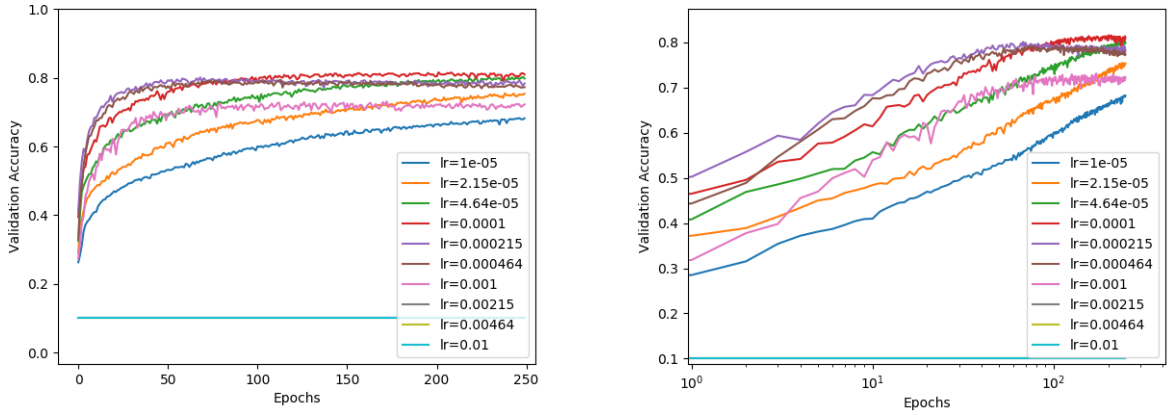


(c) Logarithmic view of the effect of learning rate



(d) A closer view of the logarithmic scale of top performing learning rates

Figure 6: Results of exploration of learning rate on MNIST



(a) The result of altering learning rate on CIFAR-10 model

(b) A logarithmic view

Figure 7: Results of exploration of learning rate on CIFAR-10

6.1.2. Determination of FGSM parameter ϵ

In the FGSM attack, the parameter ϵ determines the step size of the gradient ascent from the true classification value. Since FGSM is a single step attack, the value of ϵ becomes incredibly important to the success of the calculated perturbation. To see the effect that the value of ϵ has on the success of attacks the two test networks shown in figure 5 were used. We then experimented with the success of adversarial attacks on these models with a changing value of epsilon. From the results displayed in the graph 8, we can see that the best value of ϵ changes depending on which data set we use. The best ϵ value for an MNIST attack is 0.26, with the best value for CIFAR-10 is 0.1. From the data we can see that for the MNIST model ϵ starts out with a low success rate, then peaks at approximately 0.26, before decreasing again. This may be because as we increase ϵ we increase the step size. With low values we do not move far from the original image, leading to the poor performance in the adversarial images, and as we increase, we get closer to the ideal perturbation amount. When we exceed this value we start taking steps that are too large and miss the optimal step size, leading to worse performance. The CIFAR-10 ϵ value shows similar behaviour, shifted slightly to the left of the MNIST peak.

The peaks in the graph show that the FGSM attack has an optimal value for ϵ, which in most cases is fairly close to 0. The fairly low percentages of successful adversarial attacks on either side of the peak show the degrading image clarity, as the images become more obscured, eventually becoming noise. This can be seen quite clearly in the images in figure 9.

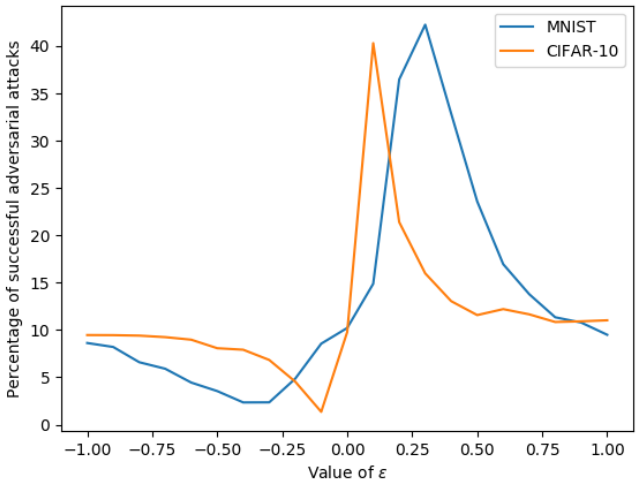


Figure 8: The effect of ϵ in FGSM on the success of adversarial attacks

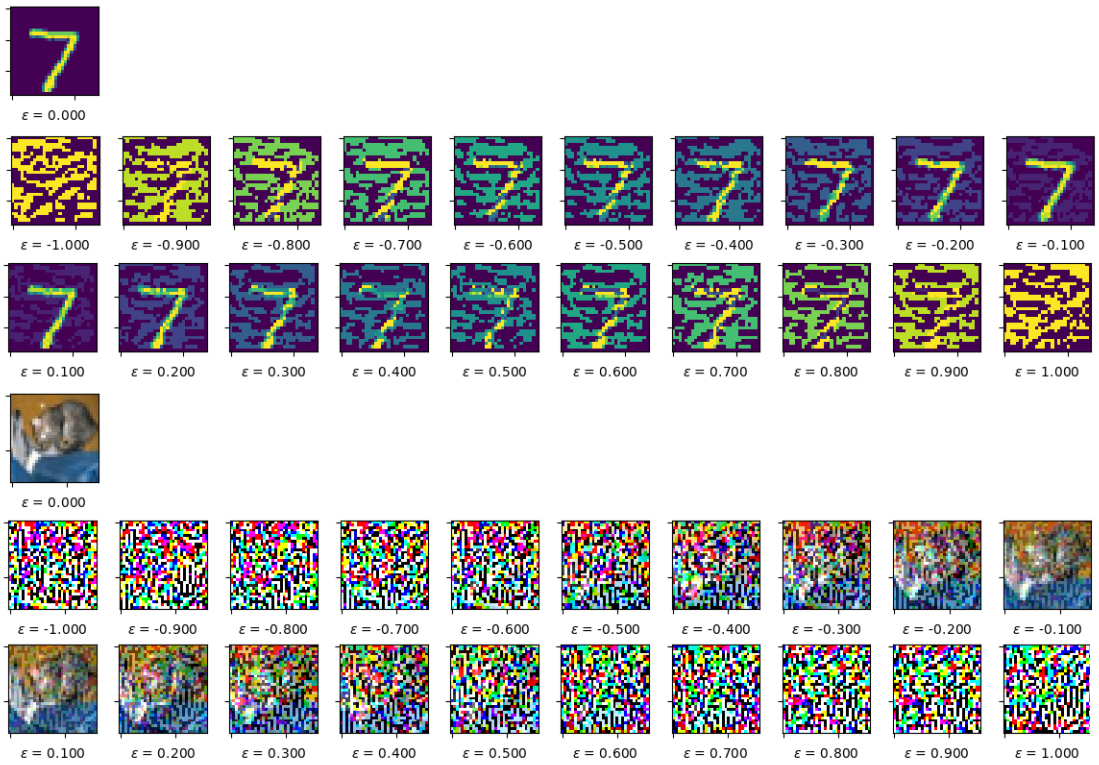


Figure 9: The effect of ϵ in FGSM on the subtlety of image perturbations

6.1.3. Determination of JSMA parameters θ, Υ

In JSMA, the parameter θ refers to the change made to the pixels, with the value, stated by Papernot et al. [20], of 1. This is also the default value provided by the CleverHans library function, therefore we started by changing Υ while setting $\theta = 1$. The value of Υ determines the maximum distortion of the image, that is, the maximum percentage of pixels modified to generate an adversarial example. To test this parameter, we ran the JSMA attack on each value of Υ from 0 to 1 with a step of 0.1 each time. The results of this can

be seen in figure 10 which shows a very high success rate in the case of the CIFAR-10 network, and a lower (but still effective enough) result on the MNIST network. To determine the best value of Υ we must also take the subtlety of the perturbation of the image into account. We can see the results of the perturbation on the images in figure 11.

The most successful value of Υ on the MNIST network is approximately 0.4 with a 75% success rate of adversarial attacks. However, from the images we can see that $\Upsilon = 0.4$ gives an obvious perturbation. The best compromise between adversarial success rate and the subtlety of attack is with $\Upsilon = 1.0$ which has a 72% adversarial success rate, whilst keeping the perturbation to a minimum. The Υ value of 1 means that all salient pixels in the image can be changed to generate an adversarial image, therefore, since we can change more pixels, we can change them by a smaller amount to get to the target classification.

The most successful value of Υ for the CIFAR-10 dataset is $\Upsilon = 0.6$ or $\Upsilon = 1.0$, which have an adversarial success rate of 99%. However, if we consult the images, the perturbations of these images are obvious. If we take a slight decrease in adversarial success rate to 98%, we can achieve a far less obvious change with $\Upsilon = 0.7$ or $\Upsilon = 0.8$.

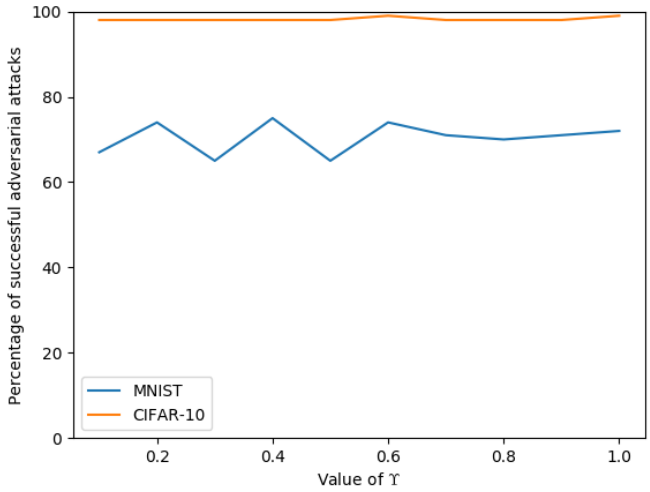


Figure 10: The effect of Υ in JSMA on the success of adversarial attacks

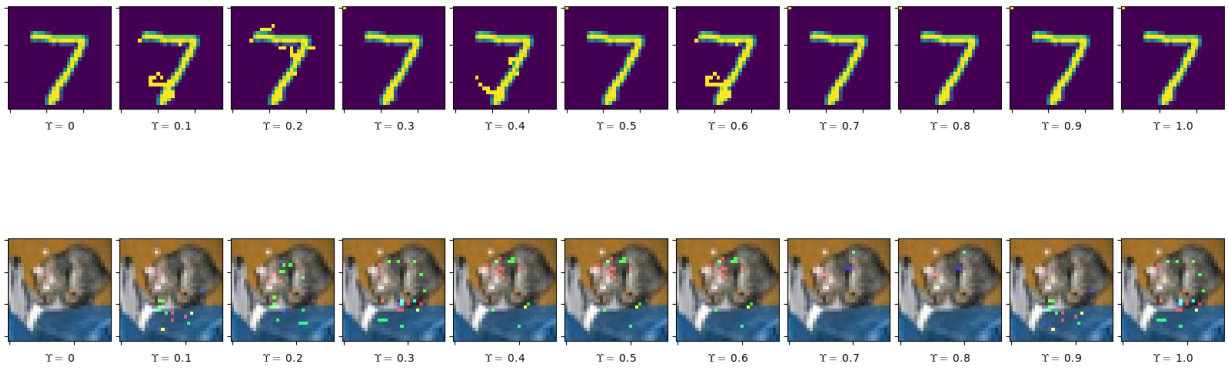


Figure 11: The effect of Υ on the subtlety of image perturbations

Due to the success of the attacks on the test networks, the decision was made to leave θ at the recommended value of 1. Since the project does not focus on making the perturbations invisible to the human eye, we can be confident that the altering of θ (which limits the maximum allowed change of a pixel) would

only provide a slight difference in the success of the JSMA attacks on the images. However, this could be explored in further work.

6.1.4. Determination of C&W L2 learning rate

With the C&W L2 attack we have the ability to alter a learning rate which can impact the effectiveness of attacks. A small learning rate leads to a more successful attack, but it is slower to converge to a good solution, therefore the balance must be struck between the two objectives of time and success rate. To evaluate the effect of learning rate on the success of C&W L2 attacks, we first look at the default value provided by CleverHans, which is 5e-3. Considering this value as a likely optimal candidate, a list of values was generated to test in the range 0 to 0.1 with the values spaced evenly on a logarithmic scale. The MNIST and CIFAR-10 models from the previous sections were then used as test networks and attacked using C&W L2 and the various learning rates. The results can be seen in figures 12 and 14.

For the MNIST network we can see from the graph that the best learning rate value is 4.64e-2, since this is the first instance of a 100% adversarial success rate. However, if we consult the images provided in figure 14, we can see that this produces an obvious perturbation of the original image. Any value over the learning rate of 4.64e-2 produces a noticeable change to the image, therefore a 100% adversarial success rate comes at this cost. If we instead reduce the success rate to 92%, we can use a learning rate of 2.15e-2, which produces an invisible image perturbation with a high rate of adversarial success.

Considering the CIFAR-10 network, the adversarial success rate quickly jumps to 100% at a learning rate of 1e-3, with this value having no noticeable impact on the original image. The close groupings of the learning rate values at the smaller end of the scale occurs due to the logarithmic distribution of values, therefore, graphs were generated to take a closer look at the smaller learning rate values 13. The graphs show that the increase in adversarial success rate is not quite as sudden as it appears on the original graph, however, finding successful adversarial attacks seems to be much easier for CIFAR-10 than for MNIST. A discussion of this can be found in the analysis section 7.1.

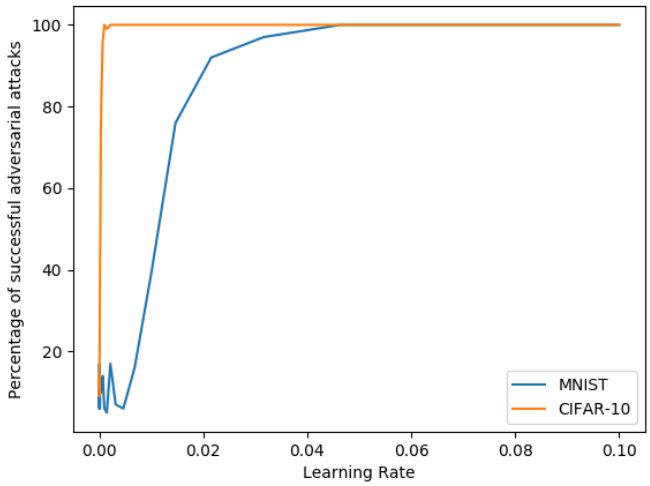
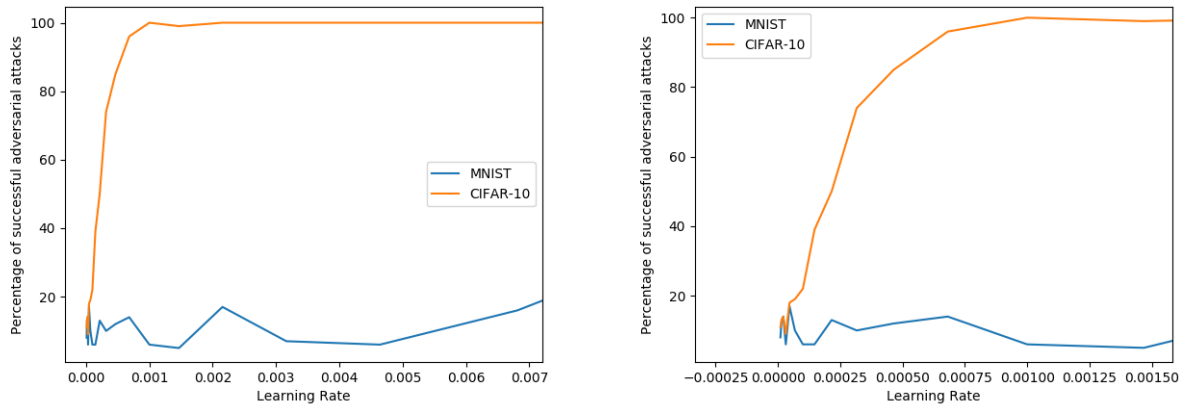


Figure 12: The effect of learning rate in C&W L2 on the success of adversarial attacks



(a) A graph of the smaller learning rates

(b) A closer view of the smaller learning rates

Figure 13: A closer look at the learning rate graph for C&W L2

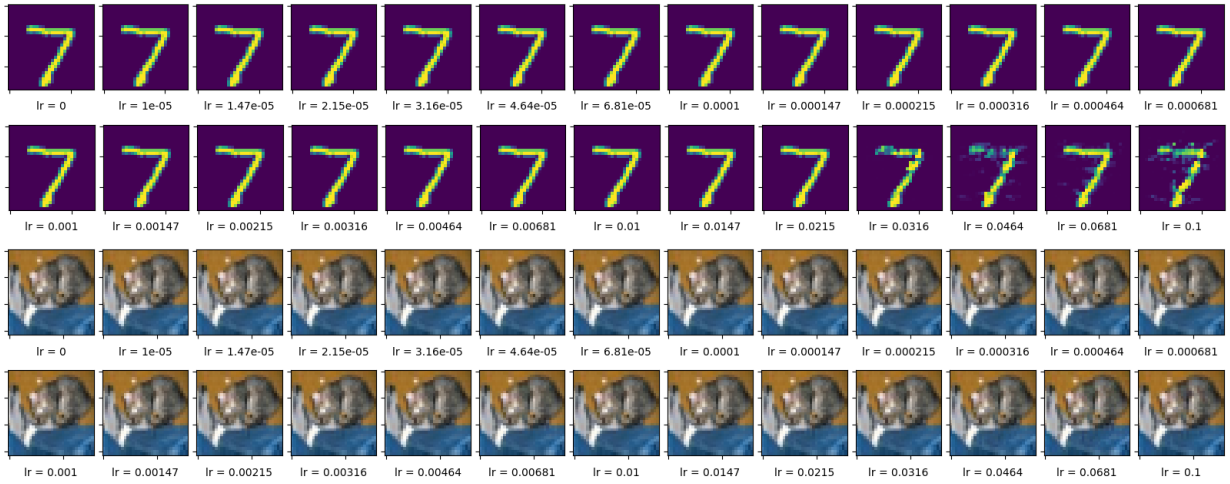


Figure 14: The effect of learning rate in C&W L2 on the subtlety of image perturbations

6.2. Generating Image Classification Networks – MNIST

The image classification generation on the MNIST dataset has been carried out with an initial population of 20 networks, and has been run for 30 epochs. Usually, it would be beneficial to have more solutions in the initial population, along with many thousand epochs. However, due to time and hardware restrictions, only a sample of the desired work can be provided.

The graphs below have an Average Value axis which refers to the average value of standard accuracy or adversarial success rate depending on the line (labelled in the legend). These values are the average values taken over all networks in the population, with the error bars at each point describing the spread of results at each epoch. Small error bars indicate a small distribution of values which show a similarity between networks in the population. This similarity can be good, since it may show success in that all of the networks can be evolved to create highly accurate and highly robust networks, but it can also indicate a high level of selection pressure, where the best solutions are consistently chosen. This produces similar networks and decreases the diversity of architectures generated. This diversity is crucial to the evolutionary process and therefore the distribution of values must be carefully monitored.

It must be noted that all of the attacks using FGSM, JSMA and C&W L2 are white-box attacks, which means they have direct access to the network to generate the adversarial images. Black-box attacks against this proposed network defence is outside of the scope of this report.

6.2.1. FGSM

Experiments were carried out by generating 20 random networks at each epoch for 30 epochs and evaluating their fitness and adversarial robustness using FGSM as the adversarial attack. This would provide a baseline to show if my evolutionary process was actually generating better networks at each epoch. The graph for this random generation can be seen as figure 15. It shows that the generation of random networks has quite sporadic behaviour in the improvement and deterioration of both the standard accuracy and adversarial success rate. This is to be expected in this situation, since there is no process occurring to guide the selection of networks, or crossover between solutions to make potentially better ones. The average standard accuracy remains at approximately 60 - 65% which is a poor standard accuracy for the MNIST dataset (human made networks easily achieve above 99% on MNIST). The adversarial success rate remains at approximately 45% throughout the epochs, which shows a poor adversarial robustness of the networks in the population. The error bars for both the standard accuracy and adversarial robustness show a deviation of approximately 15% across epochs, which shows a slight difference in the performance of networks.

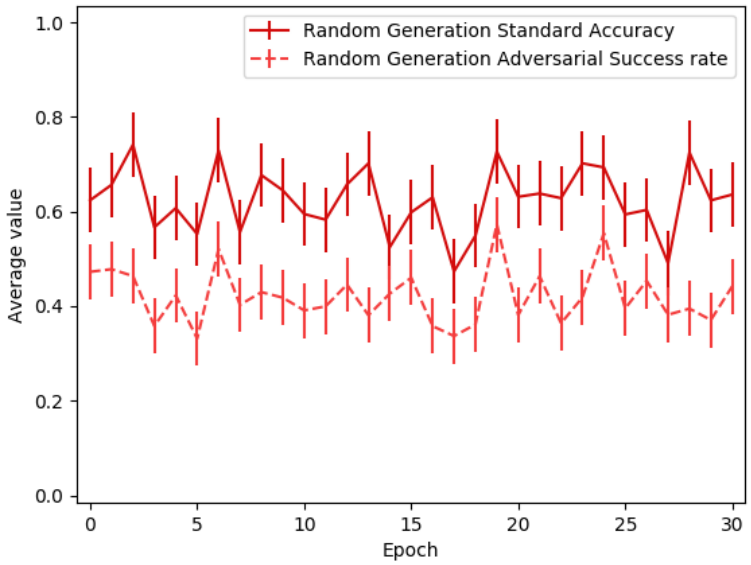


Figure 15: The random generation of networks

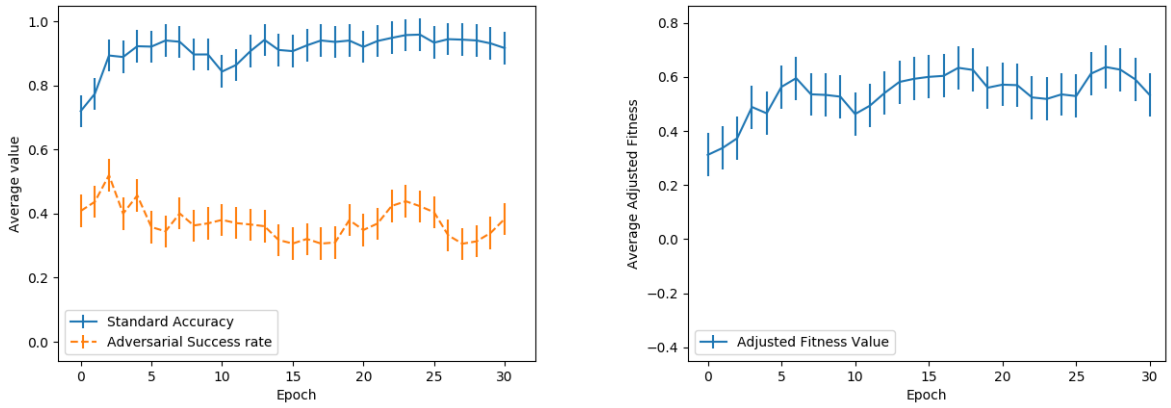
The designed evolutionary process was then used to try to improve on the random generation of networks. Starting with $\alpha = 1$ and $\epsilon = 0.26$ the fitness of the network (the percentage of correctly classified clean images) is level with the unfitness of the network (the percentage of incorrectly classified adversarial images). This means that the maximum adjusted fitness a network can achieve is 1, with the minimum being -1. This evolutionary process uses the annealing crossover process. The results for this can be seen in figure 16.

The graph (figure 16a) shows that as the epochs progress, the trend of standard accuracy increases, starting from approximately 70% and rising to close to 90% average standard accuracy. Even though this standard accuracy is below that of human made networks, the error bars indicate that the spread of standard accuracy values reaches into the high 90% with the best network over the 30 epochs (figure 17a) having an adjusted fitness value of 0.85, standard accuracy of 0.96 and adversarial success rate of 0.12. The standard accuracy of this best network is still below the best human made networks, but only slightly. Given more epochs it is possible that the standard accuracy would be able to improve to the 99% mark (provided it is possible due to the increase in adversarial robustness). The trend of adversarial success rate appears to decrease slightly over the epochs. It starts at approximately 40% and reaches a low of close to 30%. This slight decrease is encouraging and may continue in its downward trend if more epochs were used. This set of data has a spread of values of approximately 10% again and shows a small amount of diversity in the set of solutions. The decrease in adversarial success rate shows that the evolutionary process is generating

networks that are more robust to the FGSM adversarial attack. Again, this test was carried out for 30 epochs on a population of 20 networks, with 1000 FGSM images being produced to evaluate the robustness of the network.

The adjusted fitness value (figure 16b) shows a slight increase over the 30 epochs, which indicates that the evolution process is generating networks that gradually increase in adjusted fitness. The oscillation of the values between 0.4 and 0.6 is to be expected as as improvement of standard accuracy and the decrease in adversarial success rate becomes more difficult as the epochs progress.

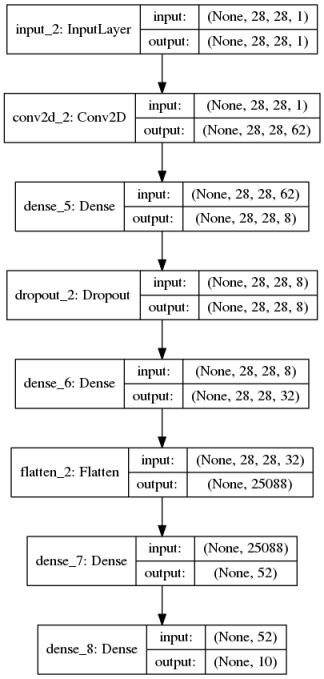
For interest the worst network was also found from the evolution process and can be seen in figure 17b with an adjusted fitness value of -0.13 (standard accuracy of 0.48 and adversarial success rate of 0.61). By comparing the two network architectures, we find that the better of the two networks has a more traditional structure which includes convolution, dense and dropout layers. The worse network contains multiple convolution and dense layers with no dropout to use for regularisation of the network. From the small sample of evolved networks that have been observed through the project, no intuitive sense regarding the correlation between a network architecture and it's robustness against adversarial attacks can be found.



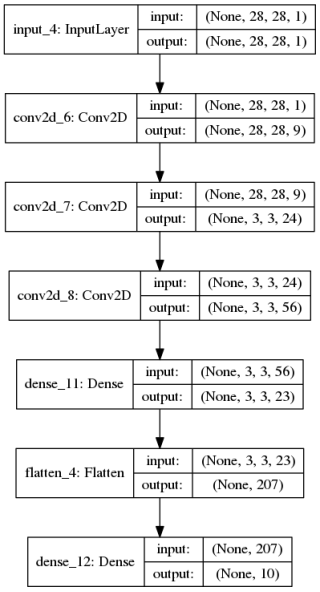
(a) The standard accuracy and adversarial success rate

(b) The adjusted fitness values

Figure 16: Evolutionary image classifier generation with $\alpha = 1$ on the FGSM attack on MNIST dataset



(a) The best MNIST network against FGSM over 30 epochs



(b) The worst MNIST network against FGSM over 30 epochs

Figure 17: The best and worst performing networks on MNIST against the FGSM attack

If we compare the data by placing them on the same graph, the difference between the random generation and evolved networks becomes clearer. This can be seen in figure 18. The graph shows that the evolutionary process usually attains a higher standard accuracy (and maintains this accuracy), whilst the evolved adversarial success rate is mostly lower than the randomly generated counterpart. The reason that the difference is not more pronounced in the adversarial success rate could be due to the lower random standard accuracy. From the tests that have been carried out, networks with a lower standard accuracy generally have a lower adversarial success rate. Therefore, since the standard accuracy of the randomly generated networks is lower, we can expect the adversarial success rates to be lower than expected.

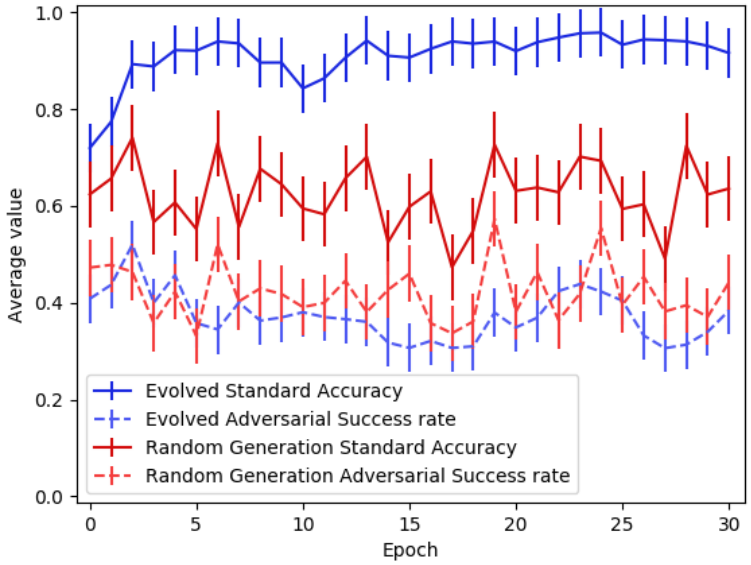


Figure 18: The comparison of evolved and random networks

The performance of evolution can be proven further by using an independent two sample t-test. The null hypothesis in this test is that both sets of data we provide are drawn from the same distribution. If we provide the fitness values from the evolution and from the random generation, we should see that the null hypothesis is rejected, since the fitness values provided by the evolutionary method should be better than the fitness values provided by the random generation.

Epoch	**p-value**
21	1.97×10^{-4}
22	1.03×10^{-5}
23	2.95×10^{-3}
24	6.16×10^{-3}
25	6.88×10^{-6}
26	1.67×10^{-4}
27	5.08×10^{-7}
28	5.24×10^{-11}
29	3.49×10^{-5}
30	1.28×10^{-6}

Table 2: The p-values of an independent t-test

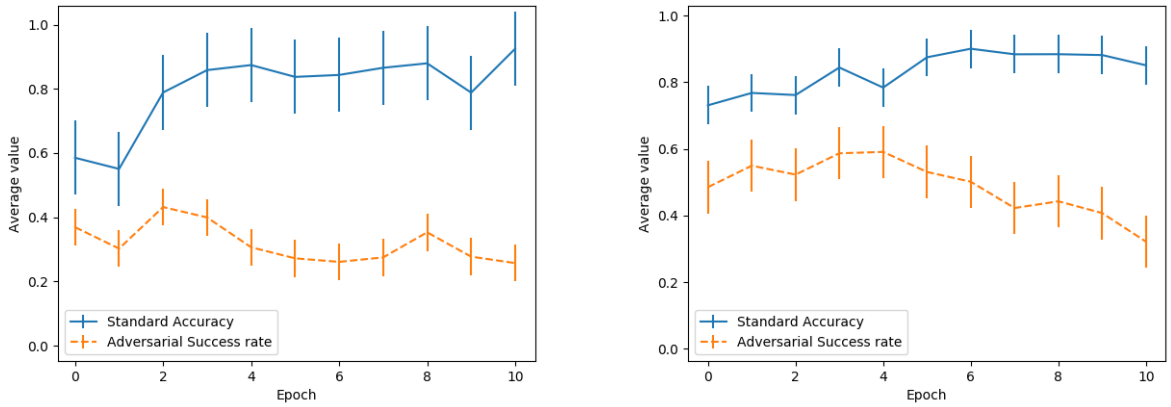
The p-values in table 2 shows the results of the independent t-tests from the last 10 epochs of the evolution run and the random generation of networks. The table shows that all the p-values are small, therefore we can reject the null hypothesis that the two datasets are drawn from the same distribution. This means that the datasets are statistically different from one another, which allows us to conclude that the evolution process is altering the fitness values in a way that is not the same as random generation. From figure 18 we can see the improvement evolution makes, therefore, we can confidently conclude that evolution is creating better networks.

6.2.2. Annealing and Random Pooling

A small experiment to compare the annealing crossover process with the random pooling crossover process was also used. This was carried out with a population of 10 networks over 10 epochs to keep the run time

as small as possible. The results are in figure 19 and show that there is little difference between the two processes. Both the random pooling and annealing methods attain a standard accuracy of approximately 85%. However, the annealing process has a smaller deviation of values from the average at approximately 10% compared to the random pooling process at close to 17%. Interestingly, this swaps when considering the adversarial success rate, with random pooling having a deviation of 10% and annealing at 17%. The annealing process shows a greater decrease in the adversarial success rate than the random pooling process. However, both processes end at about 30% adversarial success rate.

The annealing crossover process is potentially more suitable since it retains the loose structure of the network architecture. This retention of the structure ensures that dense layers cannot occur before convolution layers, which would have no effect on the network overall. This elimination of inconsequential layers leads to the production of better models and faster generation of good solutions. Had more epochs with a larger population been used, this may become more apparent.



(a) Random Pooling Crossover Method

(b) Linear Annealing Crossover Method

Figure 19: Analysis of the crossover processes

6.2.3. α values

The value of α was also experimented with to determine how it affects the generation of networks (and if it aligns with the predictions in table 1 in section 5.3.3). The results of this experiment are displayed in figure 20. The graph shows that the predictions that were made about α are correct. The evolutionary process where $\alpha = 0.25$ results in networks with a higher standard accuracy being produced, however, the adversarial success rate is also allowed to be higher and does exceed the other adversarial success rates at multiple points. If the evolutionary process was run for more epochs, it is likely that the adversarial success rate would become consistently higher for $\alpha = 0.25$ than for the other values. The plot of $\alpha = 2$ shows a lower standard accuracy, but with a generally lower adversarial success rate as well, because with a high α value, we bias towards models with lower adversarial success at the expense of standard accuracy. The value of $\alpha = 1$ is a middle-ground between the two, with a reasonable standard accuracy and adversarial success rate as the evolutionary process takes both of these factors into account equally when selecting new models. Due to the aim of creating secure networks with high standard accuracy, all evolutionary processes use $\alpha = 1$, however, an exploration of α and other fitness functions may make interesting further work.

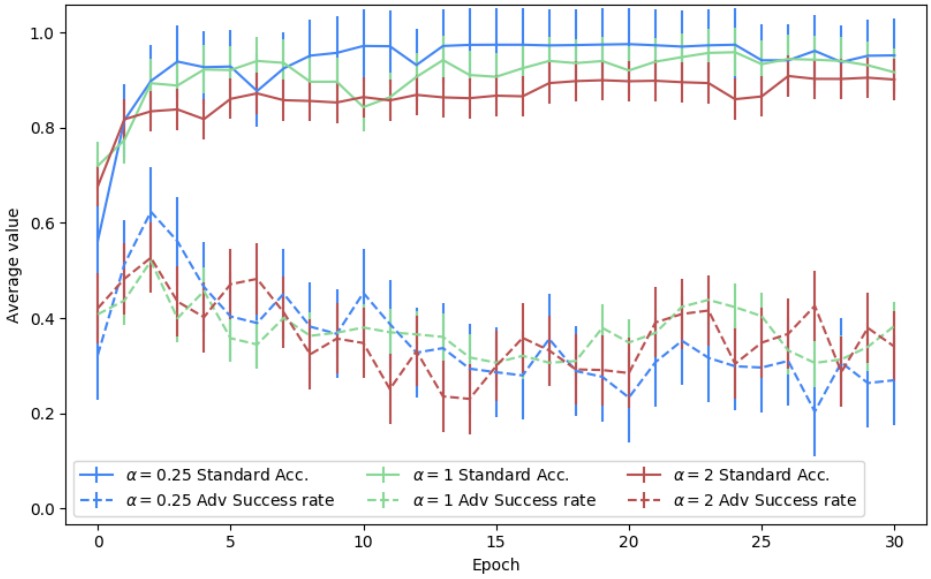


Figure 20: The comparison of α values on the network evolution process

6.2.4. JSMA

The attacks in the previous section used the FGSM attack to produce adversarial images. Now we will look at the JSMA attack.

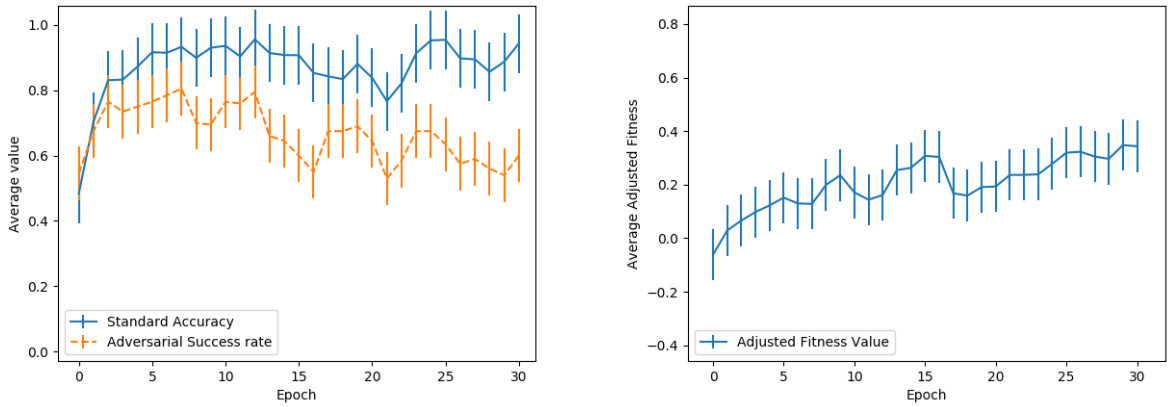
Due to time and hardware constraints, the JSMA attack tests use a slightly different set-up to the previous section. The time intensive nature of calculating the JSMA attack means that only 10 test images are used on each network to evaluate the fitness. The results of the evolutionary process ($\alpha = 1$, $\theta = 1$, $\Upsilon = 1.0$) are shown in figure 21.

The graph (figure 21a) shows that as the epochs progress the standard accuracy increases massively, from an initial standard accuracy of approximately 50% to close to 90% at the end of the run. This is a huge improvement from the initial value and shows the effectiveness of the evolution process on the ability of these networks. The slight decrease in standard accuracy (to about 80%) around epoch 20 is to be expected in evolutionary processes as the algorithm allows sub-optimal solutions to be taken over more optimal ones, since we consider both the standard accuracy and adversarial success rate in the fitness calculations used in selection. At this point we can also see a dramatic dip in the adversarial success rate which indicates that the standard accuracy values were sacrificed to find solutions with a lower adversarial success rate. The deviation of values in standard accuracy is quite high at ∼20%. This indicates that a diverse range of solutions are being considered during the early stages of this evolutionary process. The adversarial success rate shows little change through the epochs, since it starts at approximately 50% and ends at close to 60% in the final epoch. However, this is after a large jump upward in the adversarial success rate in epoch 3 (paired with a large jump in standard accuracy). Through the remainder of the epochs it appears that the evolution is fixing this initial large jump which allowed it to gain (and maintain) a large increase in standard accuracy. Had this large jump not been necessary the adversarial success rate would be expected to be much lower than shown here, and given more epochs, it could continue to fall. The variation of values in adversarial success rate is slightly less than in standard accuracy, with ∼17% variation.

The graph in figure 21b shows a clear increase in adjusted fitness value over the epochs, with a small amount of noise in the data which is reflected from the graph of standard accuracy and adversarial success rate.

The best network produced in this run can be seen in figure 22a which has a adjusted fitness of 0.91 (standard accuracy 0.91, adversarial success rate 0). This result is anomalous since a 0% adversarial success rate looks incorrect and the reason for this may be due to the small number of adversarial images used to

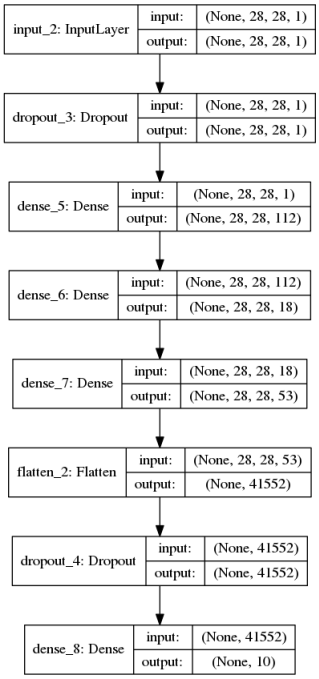
calculate the adjusted fitness value. With more time this could be confirmed by generating more adversarial images for testing. The worst network is shown in figure 22b, which has an adjusted fitness value of -0.75 (standard accuracy 0.25, adversarial success rate 1.0). Interestingly, the best network in this case consisted of no convolution layers, whereas the worst consisted almost entirely of convolution layers. Any architectural trends in robust, accurate networks is still unclear.



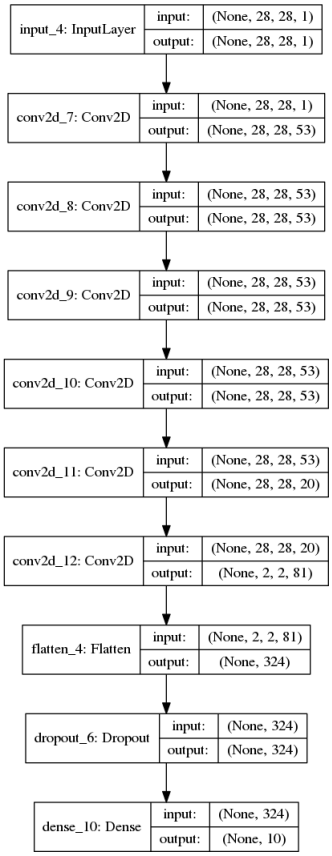
(a) The standard accuracy and adversarial success rate

(b) The adjusted fitness values

Figure 21: Evolutionary network generation against the JSMA attack on MNIST



(a) The best MNIST network against JSMA over 30 epochs



(b) The worst MNIST network against JSMA over 30 epochs

Figure 22: The best and worst performing networks on MNIST against the JSMA attack

6.2.5. C&W L2

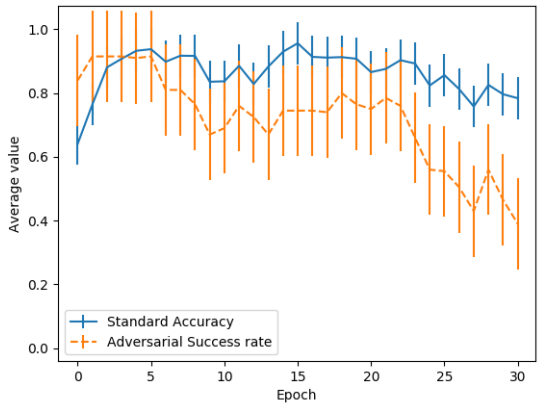
We can now look at the C&W L2 attack to analyse if the evolutionary network generation process can evolve networks that are more secure to this attack. During these tests $\alpha = 1$ and a learning rate of 2.15e-2. Similar to the tests carried out with JSMA, only 10 C&W L2 images could be produced per network to test the adversarial success rate due to the amount of time it takes to produce these images. The results of this experiment can be seen in figure 23.

In figure 23a the standard accuracy starts at close to 65% before increasing by large amounts in the first few epochs, reaching approximately 90%. There are small deviations in this value for the remainder of the run, but the general increase in standard accuracy looks promising. The small variation in standard accuracy ∼10% shows that the networks in the population have similar standard accuracy values, which may result in poor diversity of solutions. The adversarial success rate starts at about 85% success and decreases drastically to just below 40% after the 30 epochs. This is the largest decrease in the success of adversarial attacks seen so far, with the result validating evolution as a potential method of defence against these attacks. The spread of values at each epoch for this plot is quite large at over 20%. This shows a large spread of adversarial success rates in the population which indicate the diversity of solutions with respect to this variable.

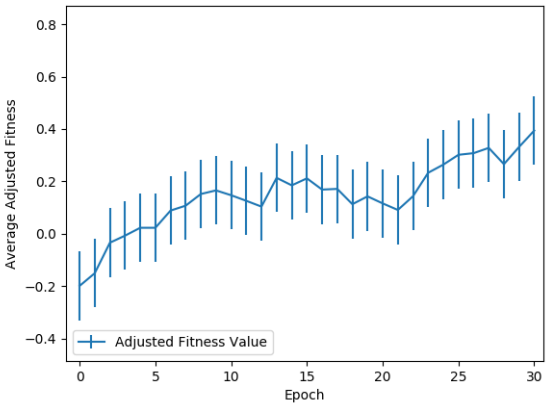
The graph of adjusted fitness values (figure 23b) shows a clearly increasing trend over the 30 epochs and indicate the success of the evolution method on this attack type.

The best network produced in the 30 epochs can be seen in figure 24a which has a adjusted fitness of 0.88 (standard accuracy 0.98, adversarial success rate 0.10). The worst network is shown in figure 24b, which has an adjusted fitness value of -0.84 (standard accuracy 0.16, adversarial success rate 1.0). The best network seems to almost entirely consist of convolution layers, as does the worst network. The difference between the

two is the number of convolution layers and a single dropout layer in the better network. This again gives no insight into the architecture of networks and the effect on adversarial robustness.

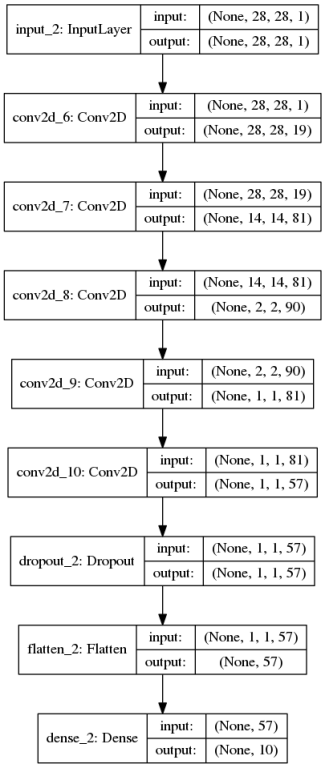


(a) The standard accuracy and adversarial success rate

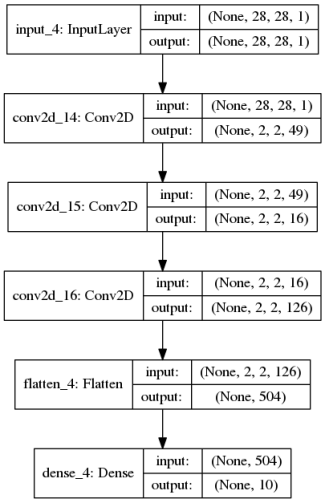


(b) The adjusted fitness values

Figure 23: Evolutionary network generation against the C&W L2 attack on MNIST



(a) The best MNIST network against C&W L2 over 30 epochs



(b) The worst MNIST network against C&W L2 over 30 epochs

Figure 24: The best and worst performing networks on MNIST against the C&W L2 attack

6.3. Generating Image Classification Networks – CIFAR-10

So far, we have only considered attacks on the MNIST generated networks. In the next few sections, we will explore the success of adversarial attacks on CIFAR-10 generated networks. Similar to the MNIST

tests above, a population of 20 networks were used. However, due to the increase in time to train a network on the CIFAR-10 dataset, only 10 epochs were used rather than the previous 30. Just as before in the MNIST graphs, the Average Value axis refers to the average of all the members of the population, with the meaning described by the line (labelled in the legend). The error bars indicate the spread of results over the population, which indicates the diversity of members with respect to the stated metric.

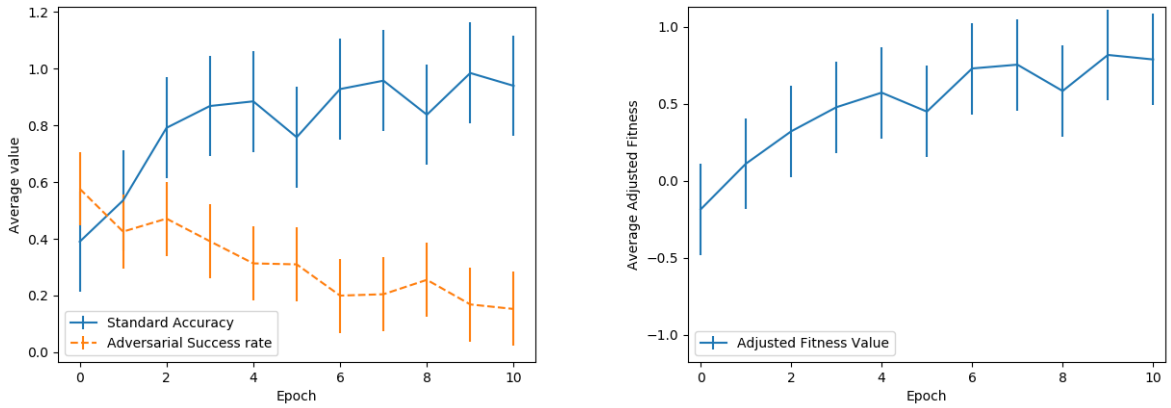
6.3.1. FGSM

The graph in figure 25 shows the results of evolving the population of networks against the FGSM attack. The generation used $\alpha = 1$ and the FGSM attack used $\epsilon = 0.1$.

In figure 25a, the standard accuracy starts low at the 40% mark, but quickly rises to finish at ∼90%. The trend in the data is clearly increasing before levelling off which is to be expected as we reach the 90% mark. However, the variation of about 30% in the average values is higher than any of those seen in the previous section which shows that the standard accuracy values in the population vary by large amounts. This indicates that the population is diverse. However, since it is small, we risk not being able to make progress to an optimal solution since the solutions are spread too thinly over the search space. In practice, this would lead to a stall in both the maximum standard accuracy achieved and the decrease in adversarial success rate. The adversarial success rate shows a clear decrease as the evolution process continues. Starting at close to 60%, it finishes at just below 20% which strongly indicates the effectiveness of the evolution in reducing the adversarial success rate. The error bars show a variation of about 20% which (from the previous runs) seems fairly standard. The difference in spread of values between the adversarial success rate and standard accuracy shows that the population is less diverse with respect to the adversarial success rate, which could indicate it is closer to an optimum in this search space.

The adjusted fitness values (figure 25b) show a strong increase over the 10 epochs, but with a very high distribution of values (indicated by the large error bars). This indicates that the population is very diverse, and may be too diverse to converge to a good solution in the future.

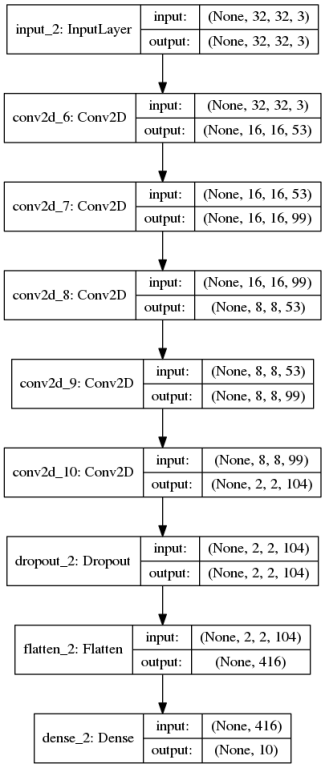
The best network produced in the 10 epochs can be seen in figure 26a which has a adjusted fitness of 0.90 (standard accuracy 0.99, adversarial success rate 0.09). This network has a suspicious standard accuracy of 99% which if true, would exceed the current state of the art in image classification on the CIFAR-10 dataset. The worst network is shown in figure 26b, which has an adjusted fitness value of -0.90 (standard accuracy 0.09, adversarial success rate 0.99).



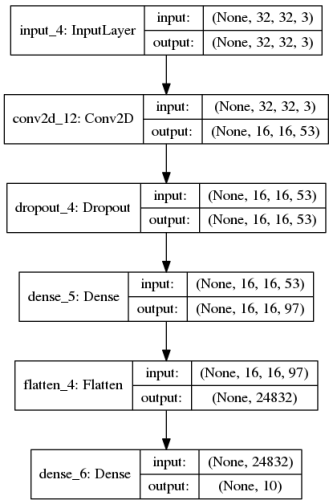
(a) The standard accuracy and adversarial success rate

(b) The adjusted fitness values

Figure 25: Evolutionary network generation against the FGSM attack on CIFAR-10



(a) The best CIFAR-10 network against FGSM over 30 epochs



(b) The worst CIFAR-10 network against FGSM over 30 epochs

Figure 26: The best and worst performing networks on CIFAR-10 against the FGSM attack

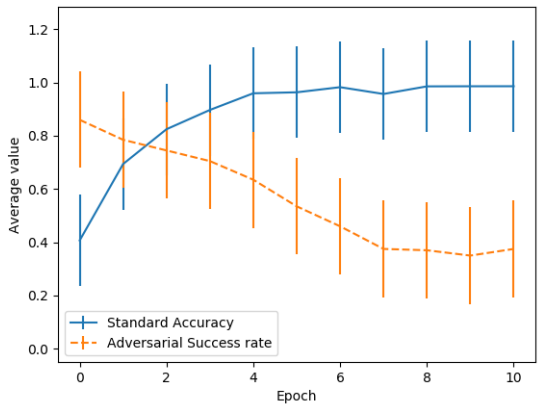
6.3.2. JSMA

Figure 27 shows the effect of the evolutionary process on the success of the JSMA attack on networks ($\alpha = 1$, $\theta = 1$, $\Upsilon = 0.7$).

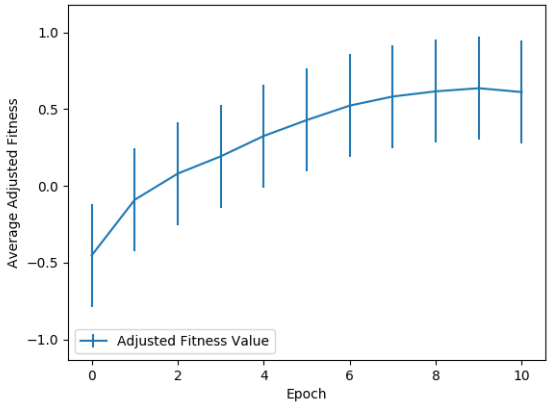
The standard accuracy plot (figure 27a) shows the increase from the initial random generation of just over 40% average standard accuracy to about 90% at the end of the evolution process. The value of standard accuracy reaching above the 80% mark also indicates that evolution is producing better networks than those created by hand, with the network used for testing the hyperparameters of the attacks only reaching an accuracy of ∼80% (see table 5). The error bars show a spread of values of just under 40% which is the largest variance seen so far. This shows a clear difference in the standard accuracy of networks in the population which could lead to the inability to create and retain good solutions in the future, since if we randomly don't choose good solutions, they could be lost and this can lead to a massive decrease in average standard accuracy. The adversarial success rate on the same graph shows a clean and defined decrease from an initial value of approximately 85% to close to 40% at the final epoch. This is a strong decrease in the effectiveness of JSMA against networks in the population and hints at the success of evolution as a possible defence mechanism. The error bars show a variance of close to 40%, similar to the standard accuracy, which may present challenges in later epochs as described above.

The adjusted fitness values (figure 27b) show an unwavering increase with a high distribution of values (similar to the previous experiment). This high distribution of values is concerning and may be due to the massive search space that must be covered during evolution.

The best network produced in the 10 epochs can be seen in figure 28a which has a adjusted fitness of 0.99 (standard accuracy 0.99, adversarial success rate 0.0). This again shows suspicious results with an abnormally high standard accuracy and a very low adversarial success rate. This would need to be verified using further tests on the network. The worst network is shown in figure 28b, which has an adjusted fitness value of -0.66 (standard accuracy 0.34, adversarial success rate 1.0). This worst network contains no convolution layers and has a single dense layer among average pooling and dropout layers.

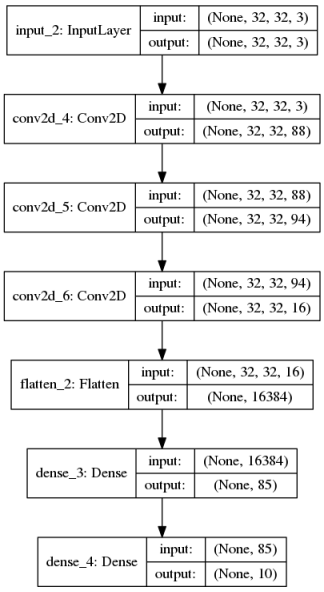


(a) The standard accuracy and adversarial success rate

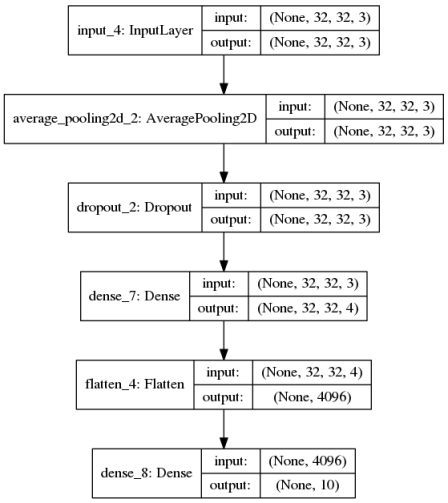


(b) The adjusted fitness values

Figure 27: Evolutionary network generation against the JSMA attack on CIFAR-10



(a) Best CIFAR-10 network against JSMA over 30 epochs



(b) Worst CIFAR-10 network against JSMA over 30 epochs

Figure 28: The best and worst performing networks on CIFAR-10 against the JSMA attack

6.3.3. C&W L2

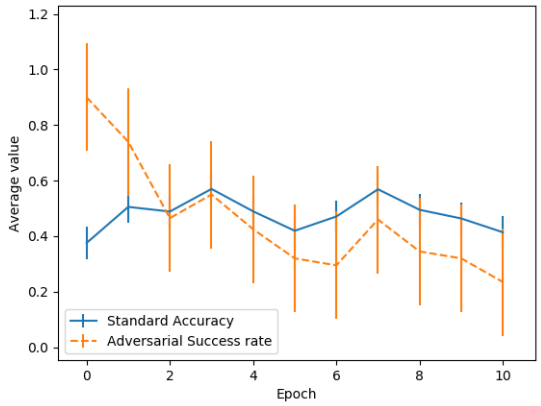
This test was carried out with $\alpha = 1$ in the evolutionary process and a learning rate of 1e-3 for the C&W L2 attack. The first set of results are shown in figure 29a and 29b.

The graph in figure 29a shows a surprisingly low consistent standard accuracy compared to the values obtained in previous CIFAR-10 graphs, with the standard accuracy only reaching a maximum of just under 60%. This result is most likely due to a poor performing initial population of networks, which is reinforced by the error bars displayed on the standard accuracy plot showing a small 10% deviation from the mean indicating most networks had a poor standard accuracy. On the other hand, the adversarial success rate still showed a decrease from nearly 90% to about 30%, with error bars showing a spread of nearly 40% which is also concerning. The adjusted fitness value shows a good increase over the 10 epochs, however the very slow increase at just above 0.0 adjusted fitness is concerning. To check this result a new evolution was run and figures 29c and 29d were obtained.

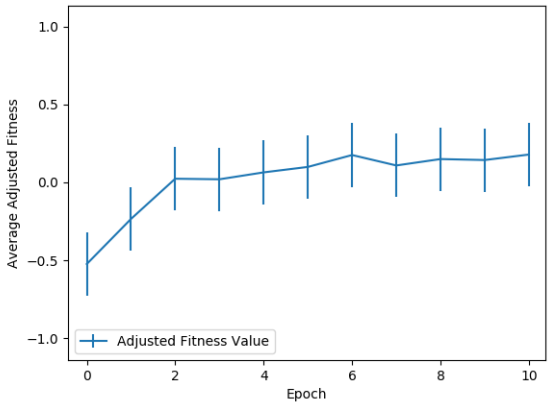
From this second run, we can see that the average standard accuracy increases quickly over the first few epochs (from about 30% to 70%). However it makes this large increase in standard accuracy at the expense of a small increase in adversarial success rate. Due to the small deviation in the value of adversarial success rates across the population, the evolution has no effective way of being able to decrease the value again, which leads to only small reductions in the adversarial success rate. The large error bars of about 40% on the standard accuracy plot also show volatility in the set of solutions with respect to standard accuracy. The adjusted fitness values (figure 29d) show little improvement above a fitness value of 0, however the distribution of values within the adjusted fitness values is much smaller than those seen in other graphs.

The evolutionary process may work better in this situation with more epochs and a larger population, since we would be able to generate more good solutions (reducing the volatility in the standard accuracy) with a greater spread of solutions with respect to the search space used for the adversarial success rate. This would enable the evolution process to generate solutions with better standard accuracy, but a reduced adversarial success rate.

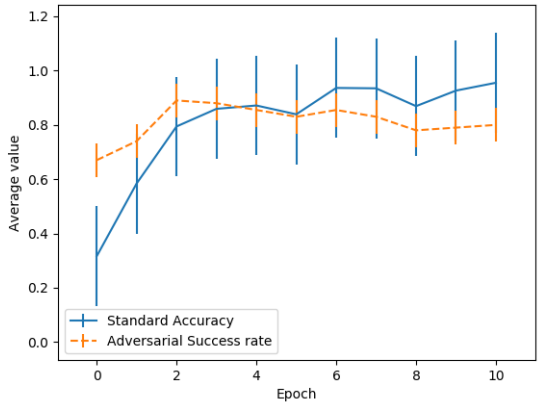
The networks displayed in figure 30 show the best and worst networks from the second run. The best network produced in the 10 epochs is in figure 30a which has a adjusted fitness of 0.68 (standard accuracy 0.98, adversarial success rate 0.30). The worst network is shown in figure 30b, which has an adjusted fitness value of -1.0 (standard accuracy 0.0, adversarial success rate 1.0). This network contains a single convolution layer followed by many dense and dropout layers. This is a poorly performing network due to the lack of convolution layers which are vital when analysing datasets such as CIFAR-10.



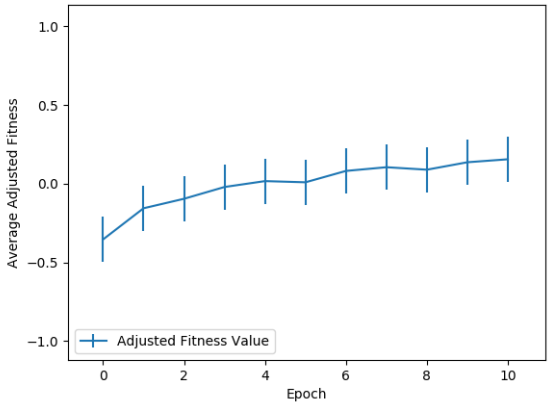
(a) Standard accuracy and adversarial success rate run 1



(b) The adjusted fitness values run 1

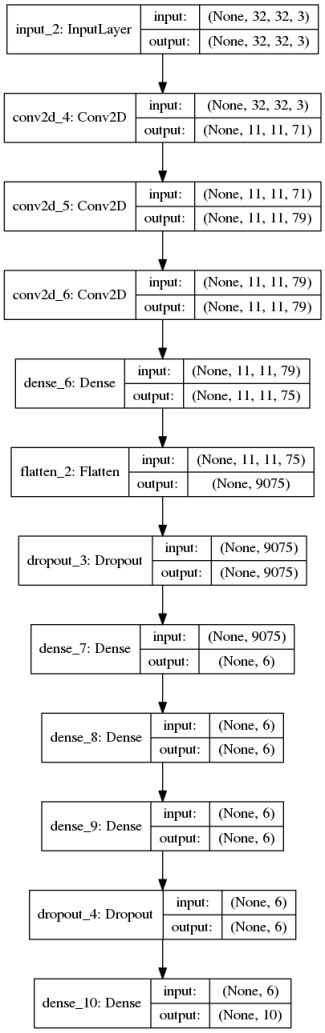


(c) Standard accuracy and adversarial success rate run 2

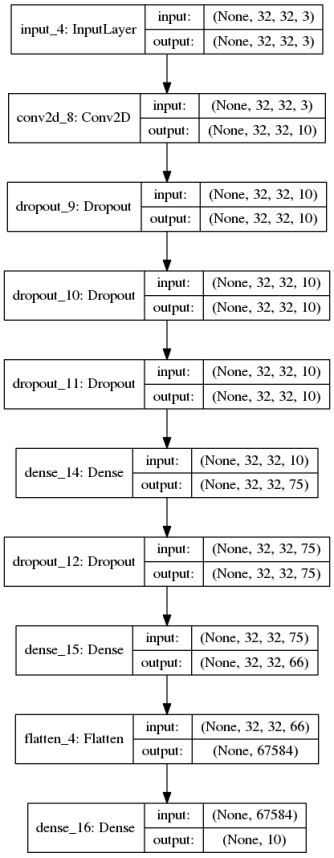


(d) The adjusted fitness values run 2

Figure 29: Evolutionary network generation against the C&W L2 attack on CIFAR-10



(a) Best CIFAR-10 network against C&W L2 over 30 epochs



(b) Worst CIFAR-10 network against C&W L2 over 30 epochs

Figure 30: The best and worst performing networks on CIFAR-10 against the C&W L2 attack from run 2

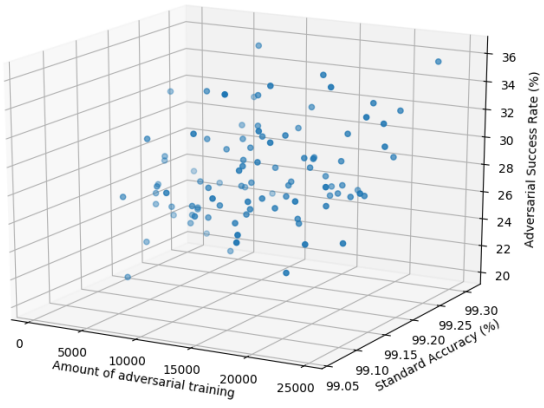
6.4. NSGA-II evaluation of defences

An aim of this project was to use NSGA-II to evaluate possible defence mechanisms to give developers greater control over trade-offs that may appear when applying defences to DNNs. This proved to be the most time intensive in terms of computer processing, therefore minimal experiments were carried out in this area.

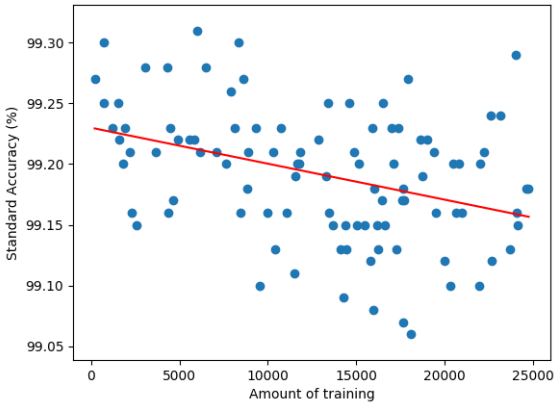
The test that was carried out looked at the adversarial training defence. This particular defence generates a number of adversarial images in a white-box attack and uses these images with the correct labels to create an augmented dataset as the images are added to the usual training set. Graph 31 shows the application of NSGA-II to this defence mechanism, where we attempt to maximise the standard accuracy, minimise the adversarial success rate and manage the amount of adversarial training that must be done. In this scenario, we can imagine an application developer comparing these three criteria, considering the standard accuracy (how usable the network is) and the adversarial success rate (how secure the network is) and how long it will take (or how expensive it may be) to train the network to this level. This last point is the amount of adversarial training. As the amount of training increases the time training takes will obviously increase, and this can lead to a higher cost if specialist hardware is used to carry out the training. In this situation, the difference in training time due to amount of adversarial training is negligible, but for larger, more complex

networks and datasets, this could take much longer, both in the training time of the network and in the production of these adversarial examples.

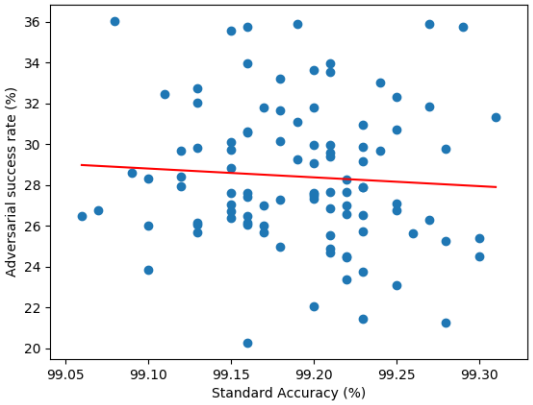
The implementation that was employed uses a small MNIST trained network (with an initial accuracy of 99.25% and initial adversarial success rate of 46.9%), and the FGSM adversarial attack. The NSGA-II algorithm selects a random amount of adversarial training to carry out in the range of 1 and 25000 images. We then generate 10000 more adversarial images on the newly trained network to get an adversarial success rate. We then evaluate the effectiveness of the defence by comparing the standard accuracy, adversarial success rate and the amount of adversarial images used during the adversarial training. The generation of adversarial images, training and evaluation takes approximately 3 minutes. The results here are from 2 iterations of NSGA-II which took 521 minutes to generate.



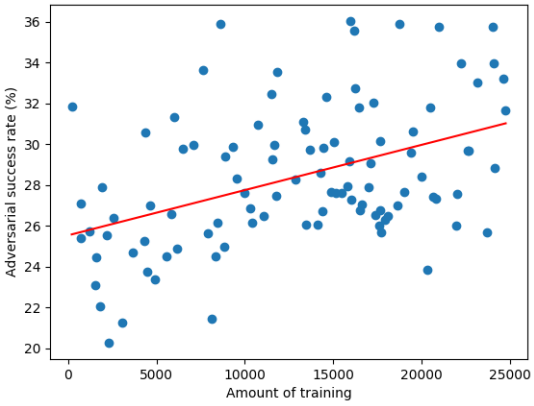
(a) A 3D view of the NSGA-II results



(b) Effect of the amount of training on standard accuracy



(c) Relationship between standard accuracy and adversarial success rate



(d) Effect of the amount of training on adversarial success rate

Figure 31: The results of the NSGA-II run evaluating the effect of different amounts of adversarial training on an MNIST network

Unfortunately, due to limited time during the project, a clear Pareto-front could not be generated. Since only 2 generations of NSGA-II could be executed, more runs may provide a more accurate and clear front.

The result in figure 31b agrees with the literature (Tsipras et al.[8]) and shows that as the amount of adversarial training increases, the standard accuracy decreases slightly. From figure 31d we can see that as the amount of adversarial training increases, the effect on the adversarial success rate is detrimental at worst, or ineffective at best. However, the increase in adversarial success rate is only slight (with an average difference of ∼4%) therefore we can conclude that it also agrees with the literature and is supported further

by the fact that *any* adversarial training decreases the adversarial success rate from 46.9% to between 36% and 20%. The graph in figure 31c indicates that there is a very weak relationship between adversarial success rate and standard accuracy, that is, as standard accuracy increases, the adversarial success rate decreases very slightly. To confirm whether this relationship actually exists, more tests must be carried out. This result does align with the predictions by Tsipras et al [8] and hints towards the tade-off between adversarial robustness and standard accuracy.

7. Analysis

7.1. Generating successful Adversarial images for CIFAR-10

The generation of successful adversarial images for the CIFAR-10 trained networks seemed to be easier than the MNIST networks for all of the attack algorithms that were looked at. This could be seen most obviously with the C&W L2 attack. Due to the discrepancy between the two networks it was decided that this would be interesting to analyse further.

If we consider the original networks of both the MNIST and CIFAR-10 models, the MNIST network had a standard accuracy of 99.31%, and the CIFAR-10 network had a standard accuracy of 81.27%. The difference in standard accuracy shows that the MNIST network has more closely approximated the ideal function for classifying hand-written digits, whereas the CIFAR-10 network (whilst having a respectable score) is more uncertain of the true data generating function. The high MNIST standard accuracy indicates that the network is able to extract the required features from the images in the dataset, which means that it generalises these features well, enabling it to achieve high standard accuracy on unseen images. Furthermore, it means that there is less uncertainty in the output space of the network, that is, it was able to approximate the complex high dimensional function well, which allows it to separate data from one class to another. Due to the complex and accurate classification boundaries the MNIST network is able to learn, this makes finding adversarial examples more difficult, since it must exploit the uncertainty in the classification boundaries (of which there is little uncertainty in this case) to be able to generate an adversarial image, especially one which has been changed in an imperceptible way.

If we then consider the CIFAR-10 network with 81% standard accuracy, we can see that it has a higher uncertainty in classification values, and therefore more loosely approximates the true data generating function and the ideal classification boundary function. This is due to the increased complexity of the dataset, where there are many more features to be extracted and generalised from the dataset to provide a good standard accuracy on new data. In essence the features are more difficult to generalise since all the images of say a cat, are slightly different from one another, colour, background etc. which means the generalisation is more difficult. This leads to a less well defined classification boundary in the output space of the network, which is further from the ideal than in the case of MNIST. This uncertainty in the classification boundaries makes it easier for an adversarial attack to move the resulting image classification over a boundary to another classification space, successfully causing a misclassification. This can be made even easier if we consider the multi-dimensional nature of the output space. This ease of moving the resulting classification also means that the perturbations to the image can be even more invisible to the human eye, by exploiting the smallest move to be made in any of the dimensions of the network output space.

Fixing this problem is not easy. The paper by Schmidt et al. [12] indicates that this problem is solved in MNIST networks since when we augment the dataset with adversarial images, we can generate enough examples to cause a strongly generalising model which can protect against adversarial attacks, however this is not an available option for datasets such as CIFAR-10 or ImageNet. The fact is that the classification tasks for CIFAR and ImageNet have such a massive variation in what constitutes a single class compared to MNIST. We can consider the example of a number "7" which is well defined (i.e. a 7 has a horizontal bar near the top of the image and a skewed vertical bar down the centre). If we then consider the class in CIFAR-10 of "bird" there are many more bird variations than the number 7, which leads to inherent noise in the classification, and the inability to create strongly generalising networks in this case.

7.2. Network Generation

A vital result regarding the evolutionary process that has been implemented is that it performs better than just random generation of networks at each epoch. This makes figure 18 a vital graph in the results section. The evolved standard accuracy on MNIST is consistently above the random generation standard accuracy (again on MNIST), which shows that the selection process that was employed (binary tournament selection) works well in selecting good solutions from the population. This is also reinforced by the (generally) lower evolved adversarial success rate compared to the random adversarial success rate.

The utility of the evolution process is confirmed by the independent t-tests performed (shown in table 2) which indicates that the evolution process is generating significantly different data than just random generation. The improvements that can be seen in figure 18 allow us to conclude that evolution provides better networks over the course of a number of epochs than the random generation of networks.

If we compare the results found from the evolution process on the CIFAR-10 dataset with the results from Real et al. [16] and Zoph et al. [17], the process in this paper was able to generate CIFAR-10 networks with average standard accuracy of 90.0% (in the worst of the runs), which is only slightly below the accuracy reported by Real (94.6% accuracy) and Zoph (97.6%). Given a greater number of epochs it is likely that these values could be matched and perhaps surpassed.

It must be noted that during the project, the efficiency of the generated networks was not considered (the number of Floating Point Operations), therefore, it would be interesting further work to compare the efficiency of the networks produced with those networks generated by Real and Zoph.

7.3. Value of α

The value of α in the fitness function $f'(x) = f(x) - \alpha\Omega(x)$ produced predictable results during the testing stage and matched well with the predictions in 5.3.3. The use of α may become more important depending on the application of a potential network. For instance, if a network is to be used in safety critical situations then a high α value would ensure that networks that are susceptible to adversarial attacks are heavily penalised and removed from the population quickly, with the obvious impact to the standard accuracy of the networks.

7.4. Generating robust networks

From the graphs on the generation and evolution of networks using both the MNIST and CIFAR-10 datasets, it would appear that this defence method is successful on these examples. Furthermore, it seems to be effective across the three attacks featured (FGSM, JSMA and C&W L2) which is an even stronger result. For all of the graphs produced, it appears that the standard accuracy shows an upward trend and plateaus at approximately the 90% accuracy mark, with the adversarial success rate decreasing gradually. The plateau of standard accuracy is to be expected since the standard accuracy is an average taken over the entire population, and therefore also accounts for the lower scoring networks that remain to improve the diversity of solutions. It is also more difficult to increase the accuracy of networks as they approach higher standard accuracy since the adversarial robustness must be balanced as well. This balance that creates the plateau could be an indicator of the proposed trade-off.

The success of the tests that have been performed show a possible network defence to consider. However, it must be noted that this is only a small test, with limited datasets, number of attacks and epochs. In addition a great amount of time was spent generating these networks, which makes the method infeasible in many situations where large complex networks are required for a task.

7.5. Comparison of Defences

We can carry out a high level comparison of the evolutionary defence and adversarial training defence by looking at the graphs produced from the evolutionary process and the NSGA-II training data, since both networks were trained on the MNIST dataset and attacked using FGSM (both with $\epsilon = 0.26$).

The evolutionary process of an MNIST network against the FGSM attack can be seen in figure 16a in section 6.2.1. The graph shows that the average standard accuracy of the networks in the population rises to approximately 90%, with the adversarial success rate reaching a low of approximately 30%, before rising up to close to 40% again. From this run, the best network in the population had a standard accuracy of 96.0% and an adversarial success rate of 12.0% (we can take the adversarial success rate as a percentage value since $\alpha = 1$).

We can compare these results to those found in the NSGA-II tests that were carried out, evaluating the adversarial training defence. The results in section 6.4 figure 31 shows the standard accuracy of the network remains high in the 99% region, which is higher than the best evolved network at 96.0%. However, the lowest value of adversarial success rate is just above 20% in the adversarial training method, which is higher than the best evolved network which attains a value of 12.0%. This indicates that the evolutionary process is able to find a more secure network, at the expense of standard accuracy (perhaps validating the Tsipras trade-off).

Furthermore, the evolutionary method allows developers to select from a population of networks which allows them to consider their individual requirements. Adversarial training on the other hand, does not allow this variety of networks to be selected.

However, we must focus on the time it takes to generate these networks using the evolutionary method proposed in this report. For an MNIST network defending against FGSM, the initial population generation takes 61 minutes, with 95 minutes per epoch after that. This is a huge amount of time compared to the adversarial training method which takes approximately 3 minutes to carry out. This time factor may even cost money if industry grade GPUs are used.

7.6. Feasibility of Evolution

From the results collected through the project, evolution seems like a practical method of generating more secure networks. However, we must consider the feasibility of this method in terms of time and required hardware. To train the image classifiers in the evolutionary process a cloud computing service was used which is specialised for machine learning research called Paperspace[2]. Using the cloud computing service, a virtual machine with an 8 core processor, 30GB RAM and an NVIDIA Quadro P5000 Pascal architecture graphics card which has 16GB RAM and 2560 CUDA Cores was used. Even with this professional grade GPU, the amount of time it took to carry out the evolution of networks was far too long to be feasible. The timings can be seen in table 3 which shows the type of run, the time taken to generate and evaluate an initial population, and the average time taken to perform all necessary crossover, mutation and evaluations of an epoch. Processes refers to the number of processes that were used for multiprocessing during evolution.

Run Type	**Initial Population Generation (Minutes)**	**Epoch Time (Minutes)**	**Processes**
MNIST – FGSM	61	95	5
MNIST – JSMA	262	76	5
MNIST – C&W L2	150	164	5
CIFAR-10 – FGSM	114	98	5
CIFAR-10 – JSMA	799	118	5
CIFAR-10 – C&W L2	151	185	5

Table 3: Time taken for generation of population and epoch for the different runs

Ideally, the evolutionary process would run for several thousand epochs, with a large population ($\sim$1000 networks). However, from the timings displayed in table 3, we can see that this would not be feasible. The runs that were carried out had a population size of 20 and took over an hour in all cases to carry out the generation of the initial population and the processing of an epoch. This would mean that a larger population and more epochs would take much longer than one hour (assuming that the amount of time scales linearly with the population size), which means that this is not a feasible method of defence.

7.7. The use of NSGA-II

The results collected using NSGA-II were anomalous in this instance, however the utility of using NSGA-II is clearly shown. By having the ability to graphically compare multiple properties of the network, we can provide application developers an invaluable method of evaluation of their models. Using this method, any potential defences can be analysed with parameters of those defences (such as amount of adversarial training) optimised automatically by the algorithm. This will then produce graphs at the end which the developer can analyse and decide the best course of action in implementing a DNN in their project. They have the ability to balance the security with the usability of a network, which is a common trade-off in the security sector. As an increasing number of DNNs are being deployed in security critical situation, it is inevitable that an evaluation method such as this will be required.

7.8. Trade-off

The analysis of the potential trade-off is difficult to carry out in this case due to the small population size and number of epochs that were used in the evolutionary process. Most the graphs produced that display

[2]Paperspace website: https://www.paperspace.com/

the adjusted fitness values (figures 16b, 21b, 23b, 25b, 27b, 29b, 29d) show a general trend of a slow increase in adjusted fitness before a decrease in gradient, or a flattening out of the values. This could indicate the presence of the trade-off discussed by Tsipras et al. [8].

The trade-off can be seen in these graphs since it indicates that we cannot improve the adjusted fitness (which takes standard accuracy and weighted adversarial success rate), past a certain point. The ideal network would have an average adjusted fitness value of 1 (a 100% standard accuracy with 0% adversarial success rate), which puts a cap on the possible values this could take. Since this perfect network should not be possible, the average standard accuracy will always be below 1. If we consider the worst possible network (standard accuracy of 0% and adversarial success rate of 100%, and $\alpha = 1$), the network will have an adjusted fitness of -1, which caps the lowest possible value at -1. Since networks realistically have a value somewhere between these bounds, a balance must be struck between standard accuracy and adversarial success rate. The work by Tsipras tells us that as adversarial robustness increases (a decrease in adversarial success rate), the standard accuracy must also decrease. The graphs of adjusted fitness values may show this due to the plateaus many of them display, as we reach the point where the evolution is unable to find an increase in standard accuracy without also increasing the adversarial success rate. Tsipras indicates that we will always reach a point where the evolution will stall since the standard accuracy cannot be improved without also increasing the adversarial success rate, which may be a trade-off that the evolutionary algorithm is not willing to take.

8. Further Work

The scope of further work on this topic is vast and a number of improvements to my work could be made including:

- More epochs
- Larger population size
- More advanced evolutionary techniques (NEAT)
- Comprehensive exploration of α and other fitness functions
- Improvements to the inclusion of NSGA-II
- A conclusion to the trade-off of adversarial robustness and standard accuracy
- Expanding to more complex datasets
- Use more advanced adversarial attacks

These are explored in the following sections:

8.1. Number of Epochs

During evolution of the networks, only a small number of epochs could be used due to time constraints and constraints over access to hardware. Ideally many thousands of epochs would be used to verify that this method can be used to generate networks that have both a high standard accuracy and good adversarial robustness. Due to the small number of epochs, we cannot get a good sense of how the evolutionary process would scale to a larger number of epochs and ultimately makes the argument for evolutionary methods of adversarial defence weaker.

8.2. Population Size

The population size for the runs that were carried out in the results section was 20, which is (in the scope of evolutionary algorithms), a very small number. This population size means that spreading over the solution space is incredibly difficult, and given the complexity and scale of the search space to begin with, only a very small amount of it was explored. This lack of exploration leads to convergence to local optima since areas with more optimal solutions are never being accessed by the population. However, the search space of the networks that were being generated was vast, and therefore some search heuristics would probably lead to better solutions than just increasing the population size.

8.3. Advanced Evolutionary Techniques

During the project a decision was made against using NEAT and NASNet as an evolutionary technique due to the lack of support from TensorFlow and Keras. However, with more time it is possible that NEAT in particular would produce better results due to the subtlety of changes to a model that could be achieved. The use of NEAT would also allow for easier comparison of different network defences. For instance, starting from a single network, we could evolve the network for a set number of epochs and evaluate it's adversarial robustness. We could then perform a different defence mechanism on the same initial network such as adversarial training. Using an evolutionary technique that considers a single network enables us to make direct comparisons with respect to other defences.

8.4. Exploration of α

The value of α in the fitness function that was used had an important impact on the models that were generated (as displayed in figure 20). A full exploration (also aided by more epochs) would more clearly show how α affects network generation in the long run, with larger values biasing networks that have higher adversarial robustness, and smaller values allowing networks to have higher standard accuracy at the expense of adversarial robustness.

It may also be beneficial to explore different fitness functions in the generation of networks. The one used in this project was fairly basic and linear, however, if (for instance) e^x was used, we could begin to produce a fitness function that allows networks with a small decrease in adversarial robustness to still be considered, whilst heavily penalising networks that have very low adversarial robustness values.

8.5. NSGA-II

NSGA-II was not used to the full extent it could have been in this project, mainly due to time constraints. The Platypus implementation of NSGA-II did not line up well with the needs of the evolutionary process. Therefore, given more time, it would be interesting to implement a custom version of NSGA-II that would be used within the evolutionary process to replace the selection process. This would ensure that good solutions are not lost (Elitism) and that the diversity of networks are being maintained, which will allow developers to decide the trade-offs they make when selecting a network (i.e. standard accuracy versus adversarial robustness).

8.6. Trade-offs

From the initial results collected, caution must be exercised when commenting on the existence of the proposed trade-off between adversarial robustness and standard accuracy proposed by Tsipras et al. [8] due to the small number of epochs that have been used. Given a larger number of epochs and population size, the relationship between the two would be clearer as, if a trade-off does exist, then networks should converge to similar fitness values as the standard accuracy cannot increase, without also increasing the adversarial success rate.

8.7. Complex Datasets

The tests that have been carried out in this project have been limited to fairly small datasets (MNIST and CIFAR-10), which are simple classification tasks. Expanding this work out to more complex datasets such as ImageNet and CIFAR-100 would be a challenge to the evolutionary process employed in this report. By testing the defence method on these larger networks, we would be able to get a more complete idea of how well this defence will work in practice, since MNIST and CIFAR-10 have very limited real world applications. Furthermore, many defences have been proposed that work well on MNIST and CIFAR-10 datasets, but fail to scale to more complex data.

8.8. Advanced Adversarial Attacks

The attacks employed (FGSM, JSMA and C&W L2) were effective at generating adversarial images for the networks that were produced. However, a diverse number of attacks exist that exploit different aspects of DNNs. For instance the Universal Adversarial Perturbation by Moosavi-Dezfooli et al. [23] uses a universal, image-agnostic perturbation vector that causes natural images to be misclassified with a high probability. This method in particular generalises well across networks, and therefore would be particularly interesting to apply to an entire population of evolved networks to compare their collective resistance using a single perturbation vector.

The testing of the evolved networks against black-box attacks such as Zeroth Order Optimisation (ZOO) created by Chen et al. [24] would also reinforce the success of this defence mechanism.

9. Conclusion

The overall aim of the project was to explore the use of evolutionary network generation in securing networks against adversarial attacks. This also included looking at a proposed relationship regarding standard accuracy and adversarial defence. Software was created to generate, evolve and evaluate populations of networks to satisfy these aims and experiment with this proposed defence. From the initial results that have been detailed, it appears that this is a possible defence which improves network standard accuracy while decreasing the success of adversarial attacks. However, it is not entirely without fault, the largest of which is its viability in its current state due to the time and resources that are required to train and evolve a population of networks. To the author's knowledge there has been no previous research in using evolutionary processes as a possible defence against adversarial attacks, therefore the results detailed in this report act as an indicator to a potentially more useful defence mechanism for securing networks as much as possible against adversarial attacks.

10. References

[1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," 12 2013. [Online]. Available: https://arxiv.org/abs/1312.6199

[2] S. Li, A. Neupane, S. Paul, C. Song, S. V. Krishnamurthy, A. K. R. Chowdhury, and A. Swami, "Adversarial Perturbations Against Real-Time Video Classification Systems," 7 2018. [Online]. Available: https://arxiv.org/abs/1807.00458

[3] W. Sultani, C. Chen, and M. Shah, "Real-world Anomaly Detection in Surveillance Videos," 1 2018. [Online]. Available: http://arxiv.org/abs/1801.04264

[4] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," 7 2016. [Online]. Available: http://arxiv.org/abs/1607.02533

[5] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust Physical-World Attacks on Deep Learning Models," 7 2017. [Online]. Available: http://arxiv.org/abs/1707.08945

[6] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter, "Accessorize to a Crime," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16*, 2016.

[7] F. Tramèr, P. Dupré, G. Rusak, G. Pellegrino, and D. Boneh, "Ad-versarial: Defeating Perceptual Ad-Blocking," 11 2018. [Online]. Available: https://arxiv.org/abs/1811.03194

[8] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry, "Robustness May Be at Odds with Accuracy," 5 2018. [Online]. Available: https://arxiv.org/abs/1805.12152

[9] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks," 11 2015. [Online]. Available: http://arxiv.org/abs/1511.04508

[10] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," 3 2015. [Online]. Available: http://arxiv.org/abs/1503.02531

[11] A. Nguyen, J. Yosinski, and J. Clune, "Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images," 12 2014. [Online]. Available: http://arxiv.org/abs/1412.1897

[12] L. Schmidt, S. Santurkar, D. Tsipras, K. Talwar, and A. Mądry, "Adversarially Robust Generalization Requires More Data," 4 2018. [Online]. Available: http://arxiv.org/abs/1804.11285

[13] E. Wong and J. Z. Kolter, "Provable defenses against adversarial examples via the convex outer adversarial polytope," 11 2017. [Online]. Available: http://arxiv.org/abs/1711.00851

[14] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," Tech. Rep. 2, 2002.

[15] K. O. Stanley and R. Miikkulainen, "Evolving Neural Networks through Augmenting Topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002. [Online]. Available: https://doi.org/10.1162/106365602320169811

[16] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, and A. Kurakin, "Large-Scale Evolution of Image Classifiers," 3 2017. [Online]. Available: http://arxiv.org/abs/1703.01041

[17] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning Transferable Architectures for Scalable Image Recognition," 7 2017. [Online]. Available: http://arxiv.org/abs/1707.07012

[18] N. Papernot, F. Faghri, N. Carlini, I. Goodfellow, R. Feinman, A. Kurakin, C. Xie, Y. Sharma, T. Brown, A. Roy, A. Matyasko, V. Behzadan, K. Hambardzumyan, Z. Zhang, Y.-L. Juang, Z. Li, R. Sheatsley, A. Garg, J. Uesato, W. Gierke, Y. Dong, D. Berthelot, P. Hendricks, J. Rauber, R. Long, and P. McDaniel, "Technical Report on the CleverHans v2.1.0 Adversarial Examples Library," 2016.

[19] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," 12 2014. [Online]. Available: http://arxiv.org/abs/1412.6572

[20] N. Papernot, P. Mcdaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Proceedings - 2016 IEEE European Symposium on Security and Privacy, EURO S and P 2016*, 11 2016, pp. 372–387. [Online]. Available: https://arxiv.org/abs/1511.07528

[21] N. Carlini and D. Wagner, "Towards Evaluating the Robustness of Neural Networks," in *Proceedings - IEEE Symposium on Security and Privacy*, 2017.

[22] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," 12 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

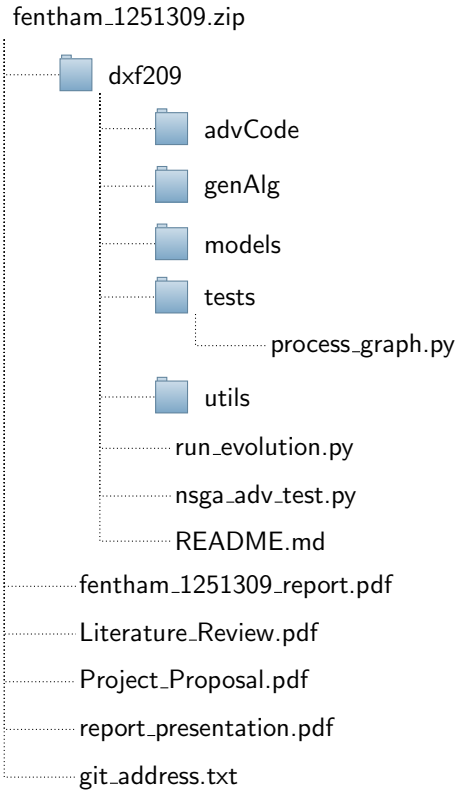
[23] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," 10 2016. [Online]. Available: http://arxiv.org/abs/1610.08401

[24] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh, "ZOO: Zeroth Order Optimization based Black-box Attacks to Deep Neural Networks without Training Substitute Models," 8 2017. [Online]. Available: https://arxiv.org/abs/1708.03999

Appendix A. Code Structure

Python was used to carry out the project (Python 3.6) with dependencies on: TensorFlow (tf-nightly-gpu 1.13.0.dev20190221)[3], Keras (2.2.4)[4], CleverHans (3.0.1)[5], Platypus (1.0.2)[6], numpy (1.16.1) and matplotlib (3.0.2).

The structure of the submission is as follows:



The folder `dxf209` contains all the code used in this report and includes sub-directories: `advCode` which contains code used to generate adversarial examples, `genAlg` contains the code used to generate, evaluate and evolve the DNNs, `models` contains the MNIST Keras model used in the NSGA-II evaluation, `tests` contains code that generates the graphs shown in the report, and contains all the data collected in the `runData` sub-directory and `utils` contains helper functions for loading datasets from Keras. The `README.md` file contains information on the project setup, the address of the repository and how to execute the main evolution code. Also included are files: `fentham_1251309_report.pdf` which is this report, `Literature_Review.pdf` and `Project_Proposal.pdf`, the literature review and project proposal carried out during the project, `report_presentation.pdf` which is the presentation for this report and `git_address.txt` contains the address for the git repository where the code can be found.

The evolutionary process can be called from the command line using:

```
python run_evolution.py
```

Which can be called with a number of command line options to alter the behaviour, these options are viewed by adding the `-h` flag.

The NSGA-II process can be called from the command line using: `python nsga_adv_test.py` Which will run the NSGA-II code and evaluate the effect of adversarial training on an MNIST network.
All code for this project can be found at: `https://git-teaching.cs.bham.ac.uk/mod-ug-proj-2018/dxf209`

[3]TensorFlow available at: https://www.tensorflow.org/
[4]Keras available at: https://keras.io/
[5]CleverHans library available at: https://github.com/tensorflow/cleverhans
[6]Platypus downloaded from: https://platypus.readthedocs.io/en/latest/

Appendix B. Project Log

Week 1 Semester 1:

Researching various topics regarding evolution of neural networks to produce models that are robust to adversarial examples. For instance, smaller networks tend to be more robust to adversarial input, therefore pairing this with an evolutionary algorithm such as NEAT would allow robust networks to be automatically created. This involves accessing and reading papers using Google Scholar and arXiv search.

Week 2 Semester 1:

Delving deeper into the methods that underpin neural evolution, including NASNet and neural fabrics. The computational power that is required for some of these worry me slightly. Also was able to get a more intuitive understanding about why smaller networks may be more robust to adversarial input. Met with my supervisor to discuss the project including the feasibility and possible directions the project can go in.

Week 3 Semester 1:

Started the project proposal. Continued to research neural evolution techniques with respect to image classification networks and discussed this with my supervisor. The main concern coming out of meetings at the moment is the amount of time and compute power these techniques will take. In response to this I have started to look at Restricted Boltzmann Machines (RBMs), and Deep Belief Networks (DBNs) which have less success than DNNs in image classification tasks, but I don't remember reading about adversarial attacks against DBNs, therefore that would be an interesting and new approach.

Week 4 Semester 1:

Completed the project proposal and began to create some image recognition networks for MNIST that work to 98% accuracy, which is 1% off state-of-the-art networks. Continued to read, looking more into other possible approaches to increasing robustness of networks, including using Restricted Boltzmann Machines and Deep Belief Networks (however the accuracy of these networks vs DNNs is a possible issue). Also began to look at a paper by DeepMind that considers how networks can know what they don't know as exploiting the assumptions networks have to make is a crucial aspect to the generation of adversarial inputs.

Week 5 Semester 1:

Spoke to my project adviser regarding my project proposal and clarified the direction of my project. Started to look at code bases which will be used in the project, which includes multi-objective NEAT which will allow an evolutionary approach to create the Pareto front which will show the trade-off between classification accuracy and adversarial robustness. This entails reading the papers that the code is based on, to understand how the code works and whether it is feasible and useful to convert this code to Python.

Week 6 Semester 1:

Found a python library (Platypus) that does multiobjective optimisation. I implemented this into some code that changes the size of a fully connected layer in an MNIST recognition system and started to create Pareto fronts which optimises accuracy and network size. Continued reading about multiobjective optimisation techniques and some alternative network styles for image recognition. Started to generate adversarial images using the Cleverhans library, which seems to be working, but visualisation of these images is proving difficult at the moment.

Week 7 Semester 1:

Implemented some code to test the Platypus library and the CleverHans library. Started working on the literature review, organising the papers I have read and the thoughts and ideas that they express. Met with my supervisor to clarify the direction of the project and the next steps that need to be taken. Continued to read papers, including one that contained the first 'wild' application of this research area in everyday life which concerns ad-blocking.

Week 8 Semester 1:

Completed literature review and created tests to ensure the Platypus library is doing what it should be. The literature states the NSGA-II is the most general and useable MOEA, therefore that will be used. Started to outline the genetic algorithm that will be used to generate networks.

Week 9 Semester 1:

Began increasing code base with an MNIST image classifier and a CleverHans FGSM implementation that can fool the network. To support the evaluation of my project I also started to write code that generates an arbitrary number of fooling images for a dataset of images and report an accuracy of how well the network

has been fooled. I'm also planning out the next stages of the project which will include the evaluation using NSGA-II.

Week 10 Semester 1:

Prepared for project inspection. Fixed some bugs in the adversarial generation code, which now generates adversaries for FGSM, JSMA and Carlini & Wagner L2 attacks. Started to implement some code for the neural evolution of networks. Had project inspection which raised some issues surrounding the amount of compute power that will be required. Hopefully the full extent of what I'll need will become clear as I conduct some tests.

Week 11 Semester 1:

Began to create the architecture for the evolutionary algorithms. Starting by implementing the two main ones focused on in the papers I have found (NASNet and NEAT) which has throw challenges such as how to structure the code to be most efficient etc. Hopefully this can give me a good idea of the computing resources I'll need next semester. I'll continue to work on this through the winter break and hopefully have some results to analyse by the start of next semester.

Week 1 Semester 2:

Worked a lot over the winter break. Started writing my report (abstract, introduction, previous work and some analysis). Wrote code for the mutations of networks via the genetic algorithm. Wrote code for adversarial training of networks and started getting results by pairing the NSGA-II algorithm with the adversarial training, comparing standard and adversarial accuracy with the amount of adversarial training done.

Week 2 Semester 2:

Implemented more advanced network generation that is more in line with the inception resnet models. Worked on the mutations of these networks, and still need to work on the crossover aspect of the evolutionary process. These evolutionary methods have been linked with the NSGA-II algorithm and some adversarial training to start producing results. Started using a more powerful machine with a good GPU to start quickly generating these networks, training and testing them.

Week 3 Semester 2:

Continued to work on network evolution and crossover (which now uses simulated annealing!). Started to run experiments on the GPU, including searches for optimal learning rates and number of epochs for the networks which is an incredibly important aspect of training. Continued to add to the report, delving into the maths behind JSMA and FGSM attacks and descriptions of how they work. I also outlined the evolutionary process in the report as well.

Week 4 Semester 2:

Found optimal learning rates and epochs for the generated models. Started to parallelise the process to make the generation and evaluation faster, which is crucial to actually getting enough data! Started to get results for the adversarial training of networks which uses NSGA-II to generate Pareto-fronts. Filled out my dissertation with some more of the maths from the attacks I'll use, and some interesting applications of these attacks to a variety of AI systems.

Week 5 Semester 2:

Started to generate results with some basic initial architectures to determine the kind of changes that occur in fooling rate with respect to the network size. Continued to add the attack maths, with FGSM and JSMA now included and (hopefully) mathematically correct. Finished a single threaded version of the generation process which is slow, but actually does what it should. Started to create classes that will allow me to train multiple models on a single GPU which Keras doesn't seem to like.

Week 6 Semester 2:

Got multiprocessing working with generating and training keras models, so the large scale evolutionary process can begin. Started generating graphs for the selection of hyperparameters and some Pareto-fronts for network standard accuracy and fooling rate against amount of adversarial training with some interesting results. Finalised the maths part of my report with adversarial attacks. Started to plot graphs of the evolutionary process vs random processes.

Week 7 Semester 2:

Started to run the network generation code for longer and collected data regarding the networks with some interesting results. Some networks have a very high accuracy ¡ 90% but also a very high fooling rate, whilst others have very low fooling rates. This leads to an interesting question about what actually causes

this difference, so I'm now hoping to do some analysis of the architectures to determine why this occurs. Continued writing my dissertation, adding the network generation process and started to add a part on the Pareto-front generation. Also generated graphs to back up my selection of hyperparameters in the generation of adversarial images.

Week 8 Semester 2:

Continued collecting results, allowing the already established programs to run for longer than I previously had. Had a conversation with my supervisor regarding the content of my report and how best to present the results I have collected. Began writing my presentation for the demonstration, which is an important part of the project since it is more research focused. Continued to add results and content to the final report, with some restructuring.

Week 9 Semester 2:

Collected and analysed more results and wrote them up in my dissertation. Created the presentation I will show during the project demo and ran the content past my supervisor. Worked on generating graphs to display the data I've collected in a cleaner and simpler way. Generated adversarial images to add to the report to illustrate these attacks. Clarified the maths aspect of the report and the presentation, and reviewed literature to make sure I can answer common questions that may come up in the demo.

Week 10 Semester 2:

Finalised presentation for my project demo, including going through referenced papers to make sure I am able to speak with confidence and (fairly) fluently on the topic. Continued data processing, collecting data for the CIFAR-10 dataset to check that the evolutionary method can scale to slightly more difficult classification problems. Continued writing report, restructuring parts and making it more readable. Re-rendered some graphs to make them clearer. Started work on responding to comments made by inspectors, so timing algorithms starting comparisons with other network defences etc.