WORCESTER POLYTECHNIC INSTITUTE

Enhancing the Robustness of Deep Neural Networks

A MAJOR QUALIFYING PROJECT

Authors:
Alasdair CAMPBELL
Jared LASSELLE

Advisor:
Dr. Yanhua LI

April 25, 2024

Abstract

Deep neural networks have demonstrated remarkable accuracy for most image classification machine learning tasks. However, these networks remain susceptible to adversarial attacks, where slight perturbations in input data produces a misclassification. Without effective defense, this vulnerability creates a significant obstacle to the practical applications of neural networks. Therefore, in this paper we propose four unique interpretations of adversarial attacks designed to test the limits of adversarial defenses. To conclude the paper we assess the strengths and weaknesses of the four defenses we designed and recommend an approach to ensure the safety and security of neural networks in the public domain.

1 Introduction

Deep neural networks (DNNs) are integral to solving many of the most challenging machine learning problems such as natural language processing, self-driving cars, computer vision, fraud detection, and speech recognition [12] [15] [3]. Therefore, ensuring the security and reliability of these systems is essential to their functionality and deployment [11]. However, research has demonstrated that even state-of-the-art neural networks are vulnerable to adversarial attack, where subtle modifications to input data can manipulate a model into an erroneous output. These alterations are often imperceptible to humans while significantly compromising the performance and safety of DNNs [3] [6] [19]. Consequently, the applications of neural networks in domains vulnerable to adversarial manipulation are severely limited unless robust defensive mechanisms can be devised to safeguard these models.

While adversarial attacks typically adhere to a common strategy of maximizing input loss while minimizing the perturbation, there is a wider variety of defensive techniques available to network architects [3] [6] [19] [11]. These strategies vary from augmenting training data with adversarial examples so a model can recognize and respond to potential vulnerabilities, to preprocessing an image in an effort to filter out adversarial perturbations [7] [2] [8]. The dynamic landscape of system security demands constant innovation in adversarial defenses as defending against adversarial attacks remains an arms race between attackers and defenders. Therefore, in this paper we assess the current state of adversarial defense mechanisms and propose alternate ways of defending a model.

In this paper we introduce:

- An initially robust and accurate neural network architecture capable of swiftly generating high-confidence predictions.
- Four distinct interpretations of adversarial attacks that exploit various vulnerabilities in neural networks to evaluate the effectiveness of our model and its defenses.

- Four adversarial defenses designed to mitigate identified vulnerabilities within our neural network architecture.
- Areas in which our attacks and defenses excel to identify potential weaknesses in existing models and areas that require more defense.

2 Preliminaries

Self-driving cars, facial recognition and social media safety filters are all real world examples of neural network applications that have a considerable impact on both general society and individual people's health and well-being [12]. Therefore, we focused our experimentation on defending and strengthening image classification neural networks. In our preliminary research we propose a convolutional neural network (CNN) with both high prediction accuracy and confidence, and outline our unique interpretations and implementations of some well-known adversarial attacks and defenses.

2.1 Convolutional Neural Networks

Numerous studies have demonstrated that convolutional neural networks are the premier architecture for tasks like image classification and computer vision. CNNs differ from standard DNNs as they are specifically designed to efficiently extract features from images. To perform this extraction a CNN leverages three fundamental layers: a convolutional layer; a pooling layer; and a linear, fully connected layer [14] [1] [10].

The convolutional layer is the cornerstone of a CNN, tasked with being the primary method of detecting features within an input. A CNN performs this operation by traversing a two-dimensional array of weights, known as a kernel, across the input to assess feature presence. Feature extraction is accomplished by computing the dot product between input pixels and the kernel weights while iteratively shifting the kernel across the input by a specified stride length until the entire input is traversed. This produces an output array known as a feature map [14] [1] [10].

Frequently following the convolutional layer in a CNN model is the pooling layer. A pooling layer employs a kernel in a similar fashion to a convolutional layer but instead of computing a dot product it aggregates all the values within the kernel, thus reducing the size of the input. This process reduces model complexity, enhances predictive speed and mitigates the risk of overfitting. In our CNN architecture we opted to use a max pooling layer, which selects the maximum value within the kernel to transmit to the output array [14] [1] [10].

Additionally, we introduced batch normalization and dropout components into our convolutional layers as these techniques are proven to regularize neural networks and expedite learning. A ReLU activation function was used between layers for its efficiency and adaptability [14] [1] [10].

The final basic element of a CNN architecture is the fully connected layer, consisting of dense, interconnected neurons that utilize the features extracted in the preceding layers to perform classifications. The network concludes with a log softmax function which computes the log probabilities of the logits generated by the output sub-layer of the fully-connected layer. It then maps them to the range of $(-inf, 0]$ to remove calculation errors frequently created by softmax functions when many values are close to 0. The largest log probability serves as the network's output for image classification [14] [1] [10].

Below, we present our complete and comprehensive CNN architecture for this research project, including the number of channels and size of the input array for each layer.

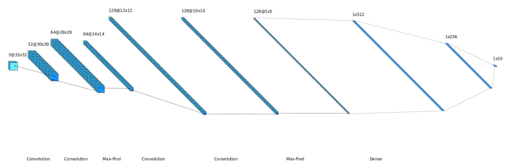


Figure 1: Implemented CNN structure. Includes two convolutional layers and one sequential layer.

2.2 Adversarial Attacks

Adversarial attacks in the context of deep learning aim to discover an alternate input, closely related to an original input, but resulting in a different output classification [19]. This scenario is frequently expressed as an optimization problem where the objective is to minimize an objective function to construct an adversarial example while maintaining its proximity to the original input [19] [3]. Or, in other words, the goal of an adversarial attack is to maximize the loss while minimizing the perturbation. This process is often formulated as [19]:

$$\begin{aligned} minimize: \quad & D(x, x+\delta) \\ so: \quad & C(x+\delta) = y' \qquad Constraint\ 1 \\ & x+\delta \in [0,1]^n \qquad Constraint\ 2 \end{aligned} \tag{1}$$

where x is the original input image, δ represents the calculated perturbation, n denotes the height and width of the image, y' signifies any class that is not the original, D depicts the distance function between the original and the adversarial inputs, and C is the classifier function. Essentially, adversarial attacks endeavor to identify the closest input to the original that causes misclassification, while adhering to the dimensions of the original image.

Solving this optimization problem typically involves defining an objective function and performing gradient descent until an optimal solution is found. However, this process is difficult and computationally expensive to solve due to the highly nonlinear nature of $C(x + \delta) = y'$. Therefore, our proposed attacks employ different and diverse methodologies to circumvent these expensive operations [19].

Approaches to solving this problem generally fall into two categories: black-box and white-box attacks [3]. Black-box attacks generate adversarial examples without needing access to internal model parameters or gradients. Alternatively, white-box attacks utilize all available information within a model, most frequently using gradients to inform perturbations. White-box attacks have been shown to be more powerful than black-box attacks, and given black-box access to a model it is possible to train an analogous model with white-box privileges [3]. However, to ensure our defenses are robust to all types of attacks, evaluating them with both white-box and black-box attacks is imperative. Therefore, in this section we outline three white box attacks (Iterative Fast Gradient Sign Method (IFGSM), Deepfool and Carlini-Wagner (CW)) and one black box attack (Iterative Pixel Swap) that will be used to evaluate our defenses.

2.2.1 Iterative Fast Gradient Sign Method

The Fast Gradient Sign Method (FGSM) is a single-step, white-box adversarial attack that prioritizes speed over creating the most visually similar adversarial image. It comprises of three main steps [6]:

1. Calculate the total loss after a forward pass.
2. Determine the gradients utilized in gradient descent during back propagation for each pixel in the input.
3. Perturb the image in the direction of the gradient that maximizes loss during forward propagation

The steps can be mathematically expressed as [6]:

$$x' = x + \epsilon * sign(\nabla_x J(\theta, x, y)) \tag{2}$$

Where x' represents the perturbed input, x denotes the original input, theta is the models parameters, y signifies the target associated with the input, $J(\theta, x, y)$ is the cost function used during back propagation, ∇_x signifies the calculated gradients, and ϵ serves as the constraining factor controlling the amount of perturbation applied to the input. Selecting an appropriate ϵ value is essential to the success of an FGSM attack, as opting for a value that is too small can result in insufficient perturbation, while too large a value can cause the perturbation to be too obvious [6].

The Iterative Fast Gradient Sign Method is an iterative extension of FGSM, where a similar algorithm is applied repeatedly until either misclassification occurs or a predetermined number of iterations is reached. IFGSM can be further optimized by integrating momentum boosting into its algorithm, which accumulates the gradients of the loss function at each iteration to facilitate escape from local maxima. These adjustments modify the initial FGSM equation to [5]:

$$\begin{aligned} x' &= x + \alpha * sign(g') \\ where: \quad g' &= \mu * g + \frac{sign(\nabla J(\theta, x, y))}{\|sign(\nabla_x J(\theta, x, y))\|} \end{aligned} \tag{3}$$

Here, ϵ retains its role of constraining total perturbation, but IFGSM introduces a new constant α which determines the perturbation step size towards ϵ, with images ultimately clipped to remain within the vicinity of ϵ. The parameter μ decays the importance of older gradients when calculating the momentum [5].

IFGSM outperforms FGSM by iteratively applying perturbations to an input, increasing the likelihood of a successful attack even if the initial iteration fails. Additionally, IFGSM mitigates the tradeoffs associated with selecting an epsilon value, as the perturbations escalate with each iteration, reducing the risks of excessively perturbing an image. Ultimately, IFGSM prioritizes speed and efficiency over an optimal input perturbation, rendering it a very effective attack algorithm, albeit potentially with more conspicuous perturbations than other techniques.

2.2.2 Deepfool

The deepfool algorithm is an iterative adversarial attack renowned for its ability to compute an optimal perturbation. Its approach involves searching neural networks for vulnerabilities by assuming the network is linear, despite the presence of nonlinear activation functions inherent to neural networks. The attack iteratively completes the following steps until the input's predicted class changes [13]:

1. Determine the closest decision boundary to the current image's classification.
2. Calculate the direction of this decision boundary from the current image's position.
3. Update the image with a small perturbation toward the selected decision boundary.

The equation used to calculate the distance to a decision boundary is as follows [13]:

$$\hat{l} = \frac{\mid f'_k \mid}{\|w'_k\|_2} \tag{4}$$

Where $\hat{l}$ is the distance to a decision boundary, f'_k are the gradients computed for class k, and w'_k is the normal vector to class k. The minimum of these distances is then used to identify the closest decision boundary, after which the gradients for that class can be computed and the perturbation calculated accordingly. The equation to calculate that perturbation is shown below [13]:

$$Perturbation = -\frac{f(x)}{\|\nabla f(x)\|_2^2}\nabla f(x) \tag{5}$$

Where f is the neural network, and x is the image in its current state. The numerator represents the initial prediction of the neural network, while the denominator denotes the magnitude of the gradient vector of the neural network squared. This fraction scales the output vector by the gradients so the perturbation is applied according to the calculated distance. The negation of this value ensures the perturbation moves the image towards the target decision boundary.

The deepfool algorithm's main benefit is its ability to compute the absolute minimum amount of perturbation required to induce a misclassification by the network. Despite the robustness of convolutional neural networks, deepfool excels at generating adversarial examples with minimal, often imperceptible, perturbations.

2.2.3 Carlini-Wagner

The Carlini-Wagner attack (CW) is a method of generating adversarial examples designed to find an optimal perturbation amount to apply to an image, but at the expense of computational speed. The core section of the CW attack utilizes two separate loss functions: one ensuring the generated image induces a misclassification, and the other determining that the perturbation is both minimal and subtly applied. As previously discussed in equation 1, the function $C(x + \delta) = y'$ is very computationally difficult to solve, so the CW algorithm reformulates the optimization problem to one that is easier to solve. This is achieved by expressing the first constraint using an objective function f so that when $C(x + \delta) = y'$, $f(x + \delta) <= 0$ [3].

In the original paper, the authors explore several objective functions and determined the most effective was:

$$f(x') = max(max(Z(x')_i : i \neq t) - Z(x')_t, -k) \tag{6}$$

Where $Z(x')$ signifies the logits for the adversarial example x', $max(Z(x')_i : i \neq t)$ represents the probability of the target class, so the expression $max(Z(x')_i : i \neq t) - Z(x')_t$ is the total difference between the model's predicted output and the desired output. By introducing a parameter k and taking the maximum value of the two expressions, we also establish a lower limit on the loss value, allowing for fine-tuning of the model's confidence on adversarial examples [3].

With this objective function defined we can reformulate the optimization problem as:

$$\begin{aligned} minimize: &\quad D(x, x+\delta) + c * f(x+\delta) \\ so: &\quad x+\delta \in [0,1]^n \end{aligned} \tag{7}$$

We introduce an additional constant c here to penalize the effect of the objective function. The optimal value of c is determined by conducting a binary search over potential values ranging from $1 * 10^{-3}$ to $1 * 10^{10}$, however the best value of c often lies between 1 and 2 [3].

This leaves us with one more constraint to solve $x + \delta \in [0,1]^n$, or the "box constraint". This problem is addressed by performing a "change of variable" operation where we optimize over a new value w rather than the over original δ. This transformation alters this constraint to [3]:

$$\begin{aligned} \delta &= \frac{1}{2}(tanh(w) + 1) - x \\ or: \quad \delta + x &= \frac{1}{2}(tanh(w) + 1) \end{aligned} \tag{8}$$

Where $tanh$ is the hyperbolic tangent, so as $tanh(w)$ varies from $[-1, 1]$, $x + \delta$ ranges from $[0, 1]$, thus solving this constraint and ensuring the values remain within the box. Therefore our final optimization problem is [3]:

$$\begin{aligned} minimize: &\quad D(\frac{1}{2}(tanh(w) + 1), x) + c * f(\frac{1}{2}(tanh(w) + 1)) \\ so: &\quad tanh(w) \in [-1, 1] \\ where: &\quad f(x') = max(max(Z(x')_i : i \neq t) - Z(x')_t, -k) \end{aligned} \tag{9}$$

The CW attack is a very effective algorithm that succeeds at a high rate and generally produces higher quality examples than other adversarial attacks. However, this efficacy comes at a computational cost, as the dual operations of a binary search and reformulating the optimization problem are computationally demanding. Therefore, CW attacks are at their most effective when computational speed and efficiency are not a concern.

2.2.4 Iterative Pixel Swap

A pixel swap attack is a concept explored in a variety of academic literature, with notable success rates demonstrated in various studies [16] [18]. For example, one study from 2022 reported a very high misclassification rate when modifying just a single pixel [16]. Our iterative pixel swap attack adheres to these fundamental principles of pixel swap attacks by:

1. Mapping pixels to each other based on a predetermined mapping function.
2. Swapping pixels according to their mapped counterparts from the previous step.
3. Repeating this swapping process until misclassification occurs.

The mapping function can be defined in a number of different ways, but our implementation focused on mapping the pixels with the most different light values. For example, the darkest pixel in the image would be mapped to the lightest pixel in the image. Subsequently, we initiate the swapping process, beginning with swapping the most dissimilar pixels until the image is misclassified.

A key advantage of a pixel swapping attack lies in its characterisation as a black-box attack, meaning it does not rely on any information gained from the network's structure or parameters. While this form of attack remains useful, particularly as models become better defended and harder to obtain information from, the perturbations produced by this attack can be extremely nonoptimal and noticeable, as there is no method to compare the closeness of the adversarial example to the original image.

2.3 Adversarial Defenses

Adversarial defenses are techniques constructed to safeguard and maintain a model's performance against attack, while having little or no effect on the model's performance on unaltered inputs [11]. In this paper we explore two distinct categories of defense: network based and image based. Network based strategies are applied directly to the neural network model, usually through further training or small changes to the architecture itself. Conversely, image based defenses manipulate the input image or associated data while leaving the original model intact. Both types of defenses have demonstrated effectiveness in minimizing the effects of adversarial examples [11] [15] [8]. However, a consequence of these defenses is they may inadvertently lower the models performance in instances where no adversarial noise is present.

2.3.1 Adversarial Examples

Adversarial example training is used to increase the robustness of a neural network by exposing it to adversarial examples during its training phase, enabling it to learn common patterns and features within adversarial examples. Typically, this technique is implemented atop an existing and high-performing model, mitigating the negative effects of adversarial example training. For our study, we took our previously defined robust network and trained it for a further five epochs using adversarial examples [7].

The most important component of adversarial example training is obtaining adequate examples. In this research project, we used the four adversarial attacks outlined in the previous section to generate our adversarial examples. Each attack perturbs images in a distinct manner, therefore exposing our model to a variety of attack techniques. We implemented this by randomly selecting and applying an attack for each batch during each pass of our training loop. By employing different attacks in each batch we prevent overfitting to specific perturbations and ensure a diverse array of attacks across epochs. For each image we retained and used its original label so the model can learn an accurate classification despite the added noise [7].

A limitation of this implementation of adversarial example training is that the model is both trained and tested using the same adversarial attacks. Consequently, we are unable to evaluate whether the model is learning general noise patterns and features, or merely those associated with each type of attack. This potentially leaves a model susceptible to a new type of adversarial attack despite exposure to adversarial examples during training. Additionally, adversarial example training extends the training time and costs necessary to produce a model beyond what is required for initial high performance. Furthermore, adversarial example training can reduce the models baseline accuracy on unperturbed images. However, regularizing a model using adversarial examples still has a positive effect on its defense, provided the impacts on its unattacked accuracy remain limited [7].

2.3.2 Feature Smoothing

Feature smoothing is the act of preprocessing the input data of a neural network to reduce the impact of noise introduced during an adversarial attack. Our implementation accomplishes this task by introducing a gaussian blur to an image, a technique that employs a kernel similarly to a convolutional or pooling layer, yet distinct in its computation of kernel weights. These weights are calculated using a normal distribution then used to aggregate values within a kernel in a similar manner to a convolutional or pooling layer. The output from a gaussian blur can be computed by [20]:

$$\sum \frac{original\ image\ value\ at\ a\ pixel\ relative\ to\ the\ original\ pixel}{kernel\ value\ for\ that\ position} \tag{10}$$

Where the kernel value is determined by [20]:

$$K(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{w\sigma^2}} \tag{11}$$

Where $K(x, y)$ represents the kernel value at (x, y), x and y are the respective distances from the current pixel being evaluated, and σ denotes the width of the kernel. This operation produces the blur by computing the weighted average of all the original values within the kernel, then replacing the target pixel with the calculated average. Consequently, small perturbations are removed as the adversarial noise is averaged out by unperturbed pixels. The resultant output image then replaces the original image as the input for the machine learning model [20].

Feature smoothing has the primary advantage of mitigating the effects of adversarial attacks by indiscriminately filtering out noise, making it a very versatile defense. It has the further advantage of requiring no further model training, meaning it is quick and inexpensive to implement. However, feature smoothing alters all images in a manner perceptible to humans, meaning its output could be perceived as inferior to defenses that do not alter images [20].

2.3.3 Gradient Masking

Gradient masking is a defensive technique primarily employed to mitigate the vulnerability of neural networks to adversarial attacks that rely on white-box access to a model's gradients. By obscuring gradient information, gradient masking aims to make a models loss function inaccurate and more difficult to optimize, therefore limiting an attack's ability to craft effective adversarial examples [2] [8].

The primary goal of gradient masking is to increase the complexity and unpredictability of a model's loss function, which impedes an adversary's attempts to leverage the gradient in an attack. There are several ways of obfuscating this gradient, including adding random noise, regularizing or normalizing the gradients, or applying some form of non-differentiable equation in the model architecture. Introducing these effects to the gradient can cause it to become noisy, flat or undefined, making the model less deterministic and harder for attacks to find a precise direction for perturbations [2] [8].

The most common approach to gradient masking requires inserting random noise on top of the gradients calculated during backpropagation. This noise perturbs the gradient information in a direction that is inconsistent with the model's parameters, therefore introducing uncertainty and inefficiency into the optimization process. This noise is typically drawn randomly from a normal distribution with a mean of 0 and a variance of 1. Additionally, to control the magnitude of the noise introduced by the gradient mask, it is multiplied by a scalar ϵ value. This parameter serves as a regularization component which constrains the noise to a reasonable level. The choice of ϵ value represents a tradeoff between robustness against adversarial attacks and preservation of model performance against clean data [2] [8].

Mathematically, gradient masking can be represented as:

$$G' = G + \epsilon * \ N(0, 1) \tag{12}$$

with G' representing the masked gradients, G representing the original gradients, ϵ depicting the scalar regularization parameter and $N(0, 1)$ defining the randomly sampled noise array.

Incorporating gradient masking into the defense of a neural network can render a model less susceptible to attack, as the perturbed gradients obscure the true loss function making it harder for an adversary to exploit network vulnerabilities. Furthermore, gradient masking is inexpensive to implement and can be incorporated into existing networks without the need for retraining. However, gradient masking may impact model performance on unperturbed inputs as a consequence of introducing this extra noise. Additionally, the benefits of gradient masking can be negated by using a black-box attack, as these attacks do not use model gradients or parameters [2] [8].

2.3.4 Defensive Distillation

Distillation as a technique was initially proposed as a mechanism to compress large neural networks into smaller ones, while maintaining the performance of the larger one. This method has since been adapted into a defensive mechanism against adversarial attacks in neural networks [15] [3].

There are three main steps to producing a distilled model [15]:

1. A large and high performing model, known as a "teacher", is trained on the original dataset using a standard approach.
2. The probabilities generated by the softmax function in the output layer of the teacher model are used to create "soft" labels, which represent the predictive distribution of classes in each input in a more detailed manner.
3. A "student" model is trained using these soft labels instead of the original, one-hot encoded targets.

Defensive distillation differs from distillation in that the goal is not to extract greater performance from a smaller model, but to defend a model against adversarial examples. As model speed and complexity are not the primary goals of defensive distillation, the teacher and student models are of equal size and structure. By distilling knowledge from the teacher to the student model, defensive distillation smooths the decision boundaries between classes, which removes some of the vulnerable regions adversarial examples frequently exploit. Furthermore, defensive distillation can combat overfitting by teaching the student model more generalized representations of classifications. However, defensive distillation is much more computationally expensive than any other of our defenses, as it requires two models of substantial size to be trained [15].

3 Methodologies

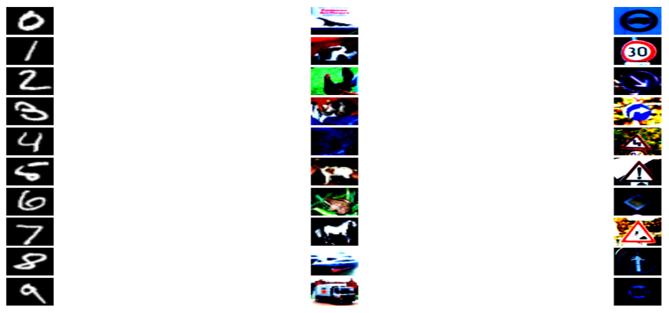
The methodologies used to complete our research can be split into five major steps. For the first step we selected three high quality datasets of differing sizes, shapes and colors. Following this, we chose three evaluation metrics that each encapsulated a different measure of model performance to assess the effectiveness of our neural networks and the corresponding attacks and defenses. Next we trained a multi layer convolutional neural network using cleaned images from the selected datasets. Finally, we developed and designed four adversarial attacks intended to exploit different vulnerabilities in neural networks and devised four defenses to protect against these attacks.

3.1 Datasets

To train a robust and precise deep neural network classifier we required large datasets complete with high quality images and labels. When a model is trained on poor quality or insufficient data, the model will perform poorly and be an inadequate base from which to evaluate our adversarial attacks and defenses. We also wished to use a variety of datasets with different features and characteristics to examine and compare our results using an array of different inputs. By using a diverse collection of datasets we were able to evaluate our results using data of varying colors, dimensions and subjects. The three datasets we chose are all popular machine learning benchmarks that fulfill these criteria: MNIST, GTSRB and CIFAR10 [17] [4] [9].

Dataset	Dataset Topic	Number of Classes	Image Dimensions and Color	Number of Images
MNIST	Handwritten Digits	10	28x28, Grayscale	60000
CIFAR10	Common Objects	10	32x32, RGB	60000
GTSRB	Traffic Signs	43	Varies, RGB	51839

Table 1: Detailed information about the contents of the MNIST, CIFAR10 and GTSRB datasets



(a) MNIST (b) CIFAR10 (c) GTSRB

Figure 2: Sample images from each of the utilized datasets

Our preprocessing plan involved normalizing all of the inputs and resizing them to become uniform 3x32x32 images. The inputs were normalized to transform all of the features to be on a similar scale to reduce the impact of outlier features. We resized the images to be 32x32 and in RGB because having consistent input size values increased compatibility with the network training process and increased comparability of our results between datasets.

3.2 Evaluation Metrics

We used two heuristics to evaluate the effectiveness of each adversarial attack: F1 Score and Prediction Time. These metrics encompass the two most important factors to consider when evaluating adversarial attacks and defenses: the accuracy of the model, and the amount of time it takes to make a prediction [21].

F1 score is a way of measuring the accuracy of a machine learning model by taking the harmonic mean of precision and recall. Precision is the proportion of positive identifications by a model that are actually positive, using the following equation:

$$\frac{TP}{TP + FP} \tag{13}$$

Conversely, recall is the proportion of actual positives that are identified positively, using the below equation:

$$\frac{TP}{TP + FN} \tag{14}$$

Precision and recall can then be combined to form an F1 score accordingly [21]:

$$\frac{2 * precision * recall}{precision + recall} \tag{15}$$

F1 score is an effective measurement of a model's accuracy as maximizing the F1 score implies maximizing both the precision and recall of the model, guaranteeing a more accurate overall network. We used the F1 score to evaluate the overall accuracy of the model and to compare the differences in scores after an attack or defense had been applied [21].

Prediction time is calculated as the amount of time it took, on average, for the model to complete one forward pass of an input. This was measured by taking the amount of time it took to test the model on the complete test dataset and dividing it by the size of the test dataset to compute the average. Prediction time is used as a benchmark to evaluate how computationally expensive an attack or defense operation is. However, it is important to note that despite conducting all tests in a consistent environment, it is possible outside variables influenced the recording of prediction time, potentially resulting in greater variability in prediction time than expected between attacks and defenses.

3.3 Convolutional Neural Network

For the purpose of introducing as few variables into our research as possible, we decided to identically implement the architecture we proposed in the preliminaries for each of our three datasets. Each dataset was divided into a train and test set in an 80:20 ratio and subsequently a further 20% of the train dataset was used to create a validation set.

We trained our models for 20 epochs, resulting in robust and effective models tailored to each dataset. Following the training phase our models generally exhibited an acceptable loss value; however it is noteworthy that the CIFAR10 dataset yielded a model with higher loss than anticipated.

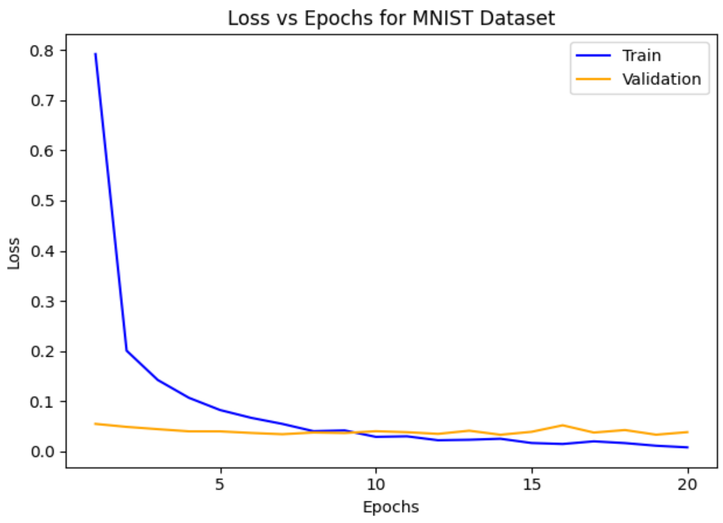


Figure 3: Training loss for MNIST. Depicts gradualy decreasing loss over 20 epochs.

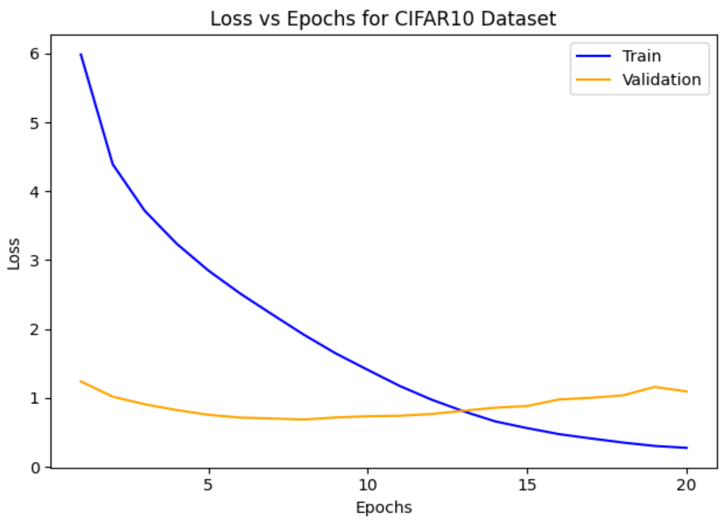


Figure 4: Training loss for CIFAR. Shows fluctuating loss over 20 epochs.

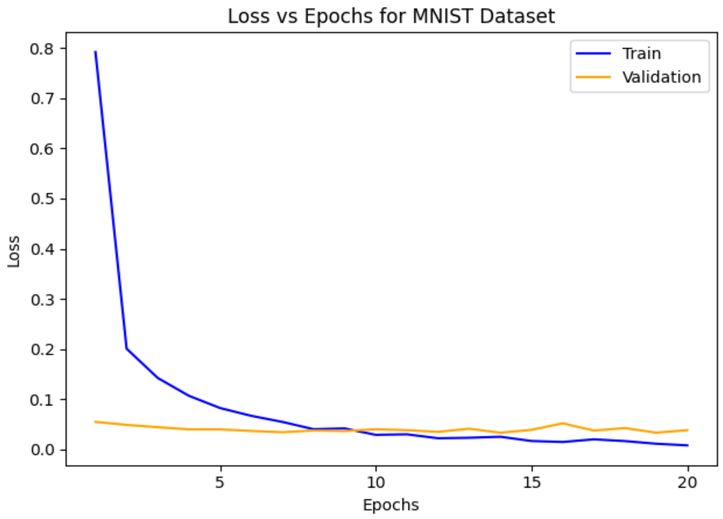


Figure 5: Training loss for GTSRB. Displays an inital rapid decrease in loss, followed by a more gradual decrease over 20 epochs.

While we trained each of our models for the full 20 epochs, to combat potential overfitting we saved the parameters of the model with the lowest validation loss, and used that model to conduct our tests. We attribute the inferior performance of the CIFAR10 model to the datasets inherent complexity; as the images have intricate and diverse features that our CNN struggled to fully capture. Constructing a model for CIFAR10 with higher accuracy would have defeated our objective of preserving a uniform network structure between datasets, and necessitated the construction of a larger and more complex model beyond the scope of this project. Furthermore, the presence of a slightly less accurate model enabled us to assess the ability of our defense methods in enhancing the baseline accuracy. Despite variations across datasets, all our models yielded satisfactory results for evaluating our attacks and defenses, characterized by high accuracy score, minimal loss, and efficient prediction times.

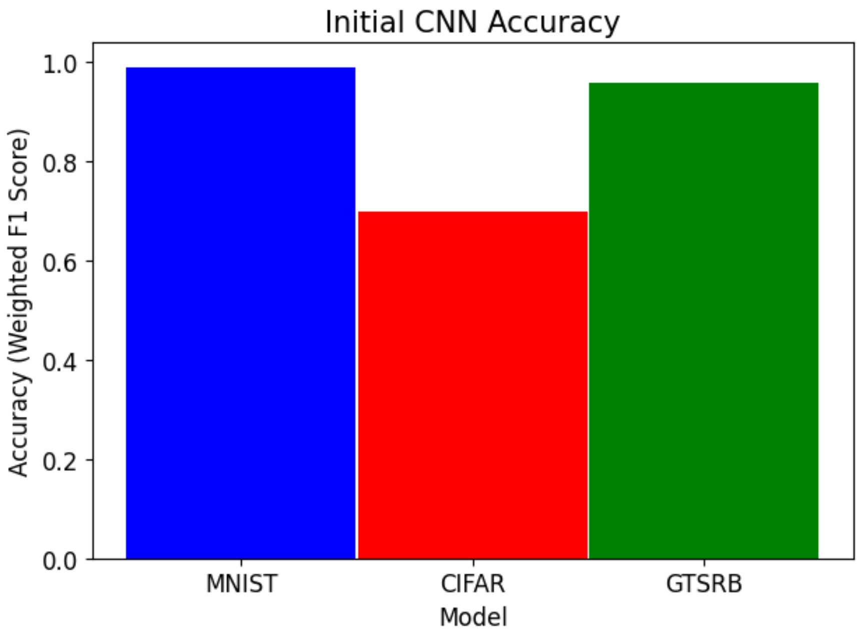


Figure 6: Baseline accuracy of CNNs. Shows at least adequate performance for each model with exceptional performance for MNIST and GTSRB

l

	MNIST	CIFAR	GTSRB
Prediction Time	0.003	0.002	0.016

Table 2: Baseline prediction time of each CNN

3.4 Adversarial Attacks

All the adversarial attacks were evaluated using the complete test dataset. We also use a digit 7 from the MNIST dataset to highlight the perturbations each of our attacks applies to an image, and the effect each attack has on the model's predictions and confidences. The baseline image and the model's 3 most confident predictions are shown below for reference.

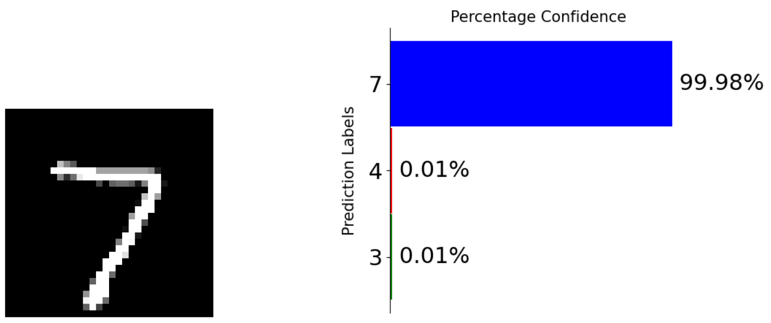


Figure 7: Baseline image and confidence of a 7 from the MNIST dataset. Shows that an undefended network classifies an unattacked image correctly with very high confidence.

3.4.1 IFGSM

The results from our IFGSM attack are as expected - it is a fast and effective attack, and this is showcased by the low F1 scores and prediction time across all datasets. This is particularly noticeable in the attack time as IFGSM is significantly quicker to perform than any of our other attacks.

l

	MNIST	CIFAR	GTSRB
F1 Score	0.08	0.01	0.11

Table 3: Accuracy of CNN after IFGSM attack. Shows poor performance of CNN on adversarial examples generated by the IFGSM algorithm.

l

	MNIST	CIFAR	GTSRB
Prediction Time	0.07	0.04	0.13

Table 4: Prediction time of CNN after IFGSM attack.

However, a surprising observation from our IFGSM attack is how little perturbation was evident on each image. IFGSM does not seek an optimal perturbation, rather, it aims to misclassify as fast as possible, so we would expect to see significant noise on each image. However, upon examining the digit 7 after it was attacked by IFGSM, there is little perturbation perceptible to the human eye. We theorize that using a small step size and momentum boosting facilitated smaller perturbations at each step, leading to a more optimal perturbation than initially anticipated. However, despite these subtle visual changes, the adversarial example significantly diverged from the original image, as the number 7 does not appear anywhere in the models three most confident predictions.

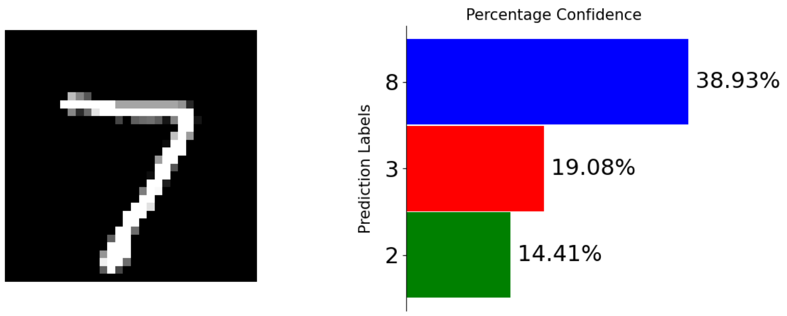


Figure 8: IFGSM attack image and confidence of a 7 from the MNIST dataset. Shows an undefended network misclassifying an IFGSM attack image with low confidence and high uncertainty.

3.4.2 Deepfool

The results from our deepfool algorithm indicate it is a very successful attack as it yielded the lowest F1 scores of any of our attacks. The attack was also performed quickly, producing a prediction time only marginally longer than for an IFGSM attack.

l

	MNIST	CIFAR	GTSRB
F1 Score	0.01	0.13	0.02

Table 5: Accuracy of CNN after deepfool attack. Shows poor performance of CNN on adversarial examples generated by the deepfool algorithm.

l

	MNIST	CIFAR	GTSRB
Prediction Time	0.21	0.19	0.41

Table 6: Prediction time of CNN after deepfool attack.

However, the deepfool algorithm frequently introduced perceptible noise to the targeted images. This noise was often manifested as a blurring of the images, which subtly altered the appearance of the image while retaining most of the prominent original features, but with a perturbation discernible by a human.

The confidences produced by the deepfool algorithm align with expectations, as the model's prediction remains very close to the decision boundary, with only a 0.01% confidence separating a prediction of a 3 to a 7. Such behavior is anticipated, as a deepfool algorithm navigates directly to the nearest decision boundary without any deviation towards other boundaries and terminates perturbing the image as soon as the boundary is crossed.

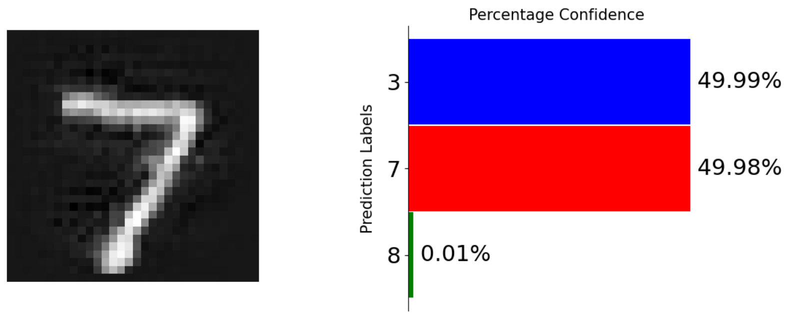


Figure 9: Deepfool attack image and confidence of a 7 from the MNIST dataset. Shows an undefended network misclassifying a deepfool attack with marginal confidence and high uncertainty.

3.4.3 Carlini-Wagner

In comparison to our IFGSM and deepfool algorithms, our CW attack exhibited relatively lower success rates and significantly longer execution times. Nonetheless, our CW attack remained a viable option for adversarial manipulation, particularly against the MNIST and CIFAR datasets. However, its performance was notably less effective on the GTSRB dataset, with an F1 score slightly below 0.5. We attribute this outcome to the combination of the complexity of the images in the GTSRB dataset and the robustness of its original model, likely causing the attack to prematurely terminate its binary search before finding an optimal constraint value.

l

	MNIST	CIFAR	GTSRB
F1 Score	0.07	0.16	0.46

Table 7: Accuracy of CNN after CW attack. Shows poor performance of CNN on adversarial examples generated by the CW algorithm.

Our CW attack was also rather slow with completion times significantly longer compared to our previous attacks, primarily due to the computational overhead introduced by the binary search for an optimal constraint value. This iterative search process necessitated creating more adversarial examples than other techniques, contributing to the overall increase in execution time.

l

	MNIST	CIFAR	GTSRB
Prediction Time	0.87	0.73	1.35

Table 8: Prediction time of CNN after CW attack

Unsurprisingly, the original and perturbed images were extremely similar. Given the primary objective of a CW attack is to identify an optimal adversarial example, it is unsurprising the perturbed images closely represent the original. Additionally, the confidence scores obtained from the CW attack closely mirror those of the IFGSM attack, suggesting underlying similarities in the operations of the two attacks.

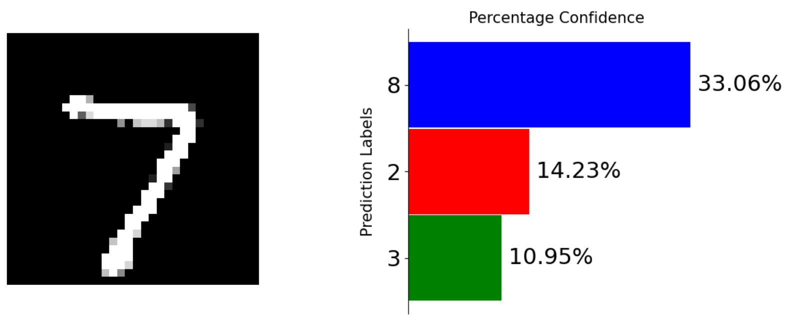


Figure 10: CW attack image and confidence of a 7 from the MNIST dataset. Shows an undefended network misclassifying a CW attack with low confidence and high uncertainty.

3.4.4 Iterative Pixel Swap

The iterative pixel swap attack emerged as the least effective of our adversarial attacks. While it exhibited a reasonably high success rate, with the highest F1 score reaching only 0.22, its overall performance remained lower than our other attacks. Furthermore, the pixel swap attack was extremely slow to operate, being significantly more time intensive than our other methods. We attribute this performance to inefficiencies prevalent in the mapping function, particularly its use of an inefficient sorting algorithm to order the lightest and darkest pixels.

l

	MNIST	CIFAR	GTSRB
F1 Score	0.11	0.20	0.22

Table 9: Accuracy of CNN after Pixel Swap attack. Shows poor performance of CNN on adversarial examples generated by the Pixel Swap algorithm.

l

	MNIST	CIFAR	GTSRB
Prediction Time	0.93	0.84	2.47

Table 10: Prediction time of CNN after Pixel Swap attack

Unsurprisingly, the resulting images from a pixel swap attack display the most noticeable deviations from the originals. This aligns with the attack's straightforward methodology of swapping pixels, a process that lacks the subtleties small changes guided by gradients can provide. Consequently, any alterations made by the attack are more conspicuous to human eyes. Additionally, the model's prediction maintains a close proximity to the decision boundary, indicating that despite a marked difference between the adversarial image and its original, the network discerns minimal differences between the two.

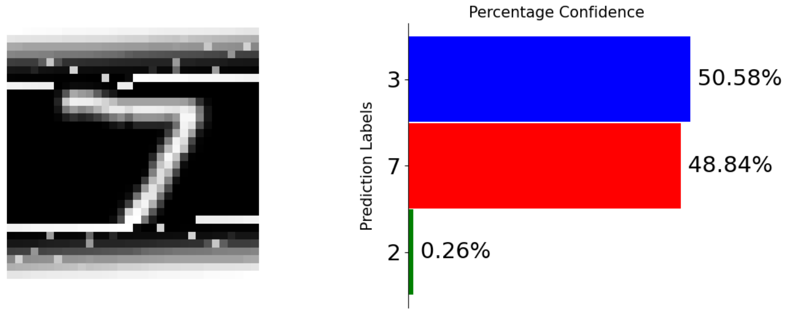


Figure 11: Pixel swap attack image and confidence of a 7 from the MNIST dataset. Shows an undefended network misclassifying a pixel swap attack with marginal confidence and high uncertainty.

3.5 Adversarial Defenses

To assess the effectiveness of adversarial defenses, we mirrored the methodology for evaluating our attacks. This approach ensured a consistent analysis framework, allowing for a direct comparison of results.

3.5.1 Adversarial Example Training

Adversarial example training proved to be an effective tool for countering adversarial attacks, demonstrating improved predictive accuracy across all attack scenarios, while maintaining performance on clean data. However, there is a notable discrepancy in the MNIST model which exhibited notably lower unattacked accuracy than its base model. This discrepancy when coupled with the remarkably high accuracies achieved on the IFGSM and CW attacks - two methods closely related to each other - suggests potential overfitting during the training process. This overfitting likely resulted in the model prioritizing the identification of adversarial noise patterns over the differentiation of images, therefore compromising its base performance.

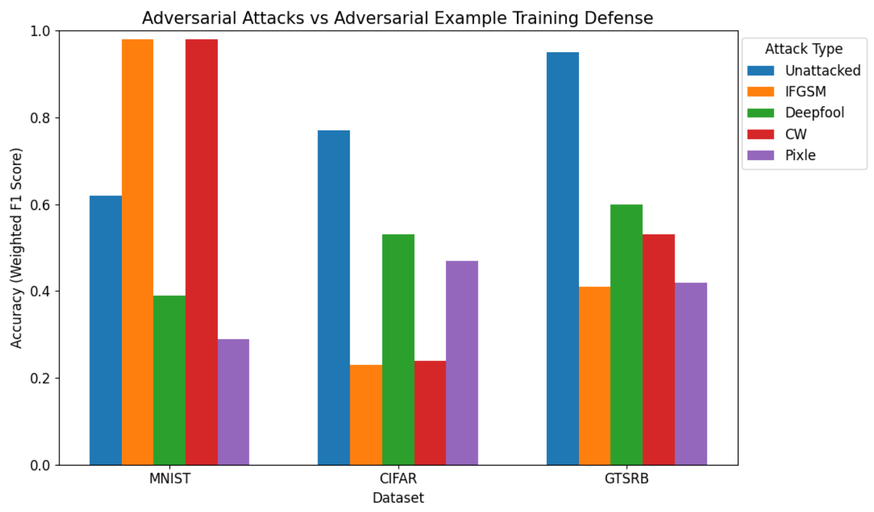


Figure 12: CNN accuracy after adversarial example training.

Despite the risks of overfitting, adversarial example training still presents numerous advantages as a defense mechanism. For instance, overfitting tendencies could be countered by training the model for less epochs or reducing the number of adversarial examples. Additionally, the prediction time was minimally impacted by adversarial training as the additional computational cost lies in the training phase rather than the inference stage.

l

Prediction Time	MNIST	CIFAR	GTSRB
No Attack	0.004	0.004	0.007
IFGSM	0.11	0.07	0.34
Deepfool	0.42	0.27	0.96
CW	0.84	0.83	1.57
Pixel Swap	0.92	0.75	2.34

Table 11: Prediction time after adversarial example training.

In conclusion, adversarial example training is a simple but effective method of mitigating the power of adversarial attacks at little cost to accuracy on clean data. Furthermore, adversarial example training is an extremely adaptable strategy that compliments other defenses well, underscoring its place as a robust defense mechanism for neural networks.

3.5.2 Feature Smoothing

Feature smoothing was overall a successful attack, with particularly noticeable effects on the deepfool and pixel swap attacks. Despite being less computationally intense to implement than other attacks and altering the images from their original look, feature smoothing was still able to improve model performance on each type of attack. We theorize that feature smoothing was particularly successful against deepfool and pixel swap as these attacks move the models confidence very close to the prediction boundary, meaning subtle changes such as aggregating pixel values have a larger effect, as the models confidence only needs to be moved a small amount to correct the prediction. We also theorize this defense does not work so well against the IFGSM and CW attacks, as those images look the most similar to the originals but have wildly different prediction confidences, meaning any noticeable changes to the image are unlikely to have such a drastic effect on those attacks.

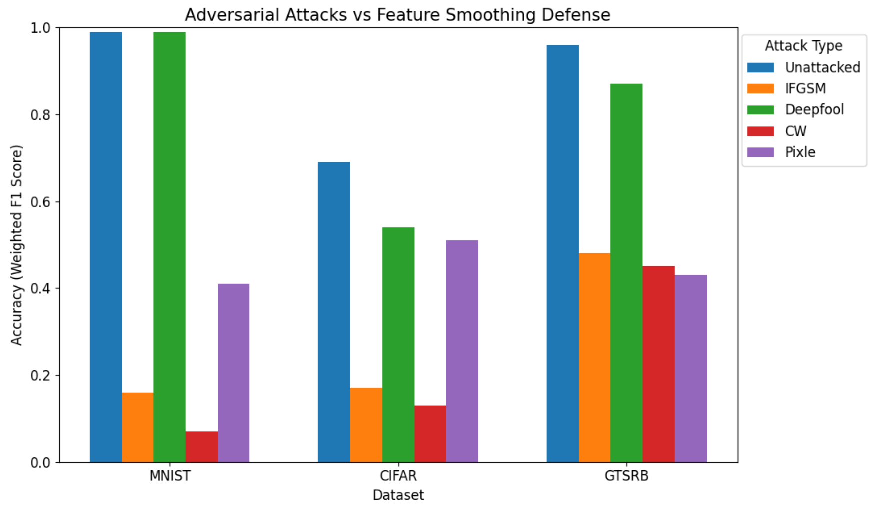


Figure 13: CNN accuracy after feature smoothing.

The impact of feature smoothing on the prediction time of our model was ultimately negligible. This result was somewhat unexpected, as feature smoothing involves a kernel sweeping across the image and performing matrix operations, which are traditionally known to be computationally demanding. However, gaussian blur techniques have been significantly optimized over time, likely explaining the speed of the operation.

l

Prediction Time	MNIST	CIFAR	GTSRB
No Attack	0.009	0.007	0.014
IFGSM	0.10	0.04	0.35
Deepfool	0.66	0.26	0.84
CW	0.73	0.65	1.86
Pixel Swap	0.81	0.75	2.76

Table 12: Prediction time after feature smoothing.

Ultimately, feature smoothing is a promising and effective form of defense against attacks strategies that perturb predictions until they lie close to the prediction boundary. However, it exhibits greater vulnerabilities than other defenses when exposed to attacks that deviate significantly from the original predictions. Therefore, its optimal use may be in combination with other defenses that mitigate this weakness.

3.5.3 Gradient Masking

Our gradient masking implementation generated disappointing results, as it exhibited little effect on classification accuracy of adversarial examples and, in some cases, even led to a decrease in accuracy on unattacked data. Notably, the prediction accuracy of our adversarial attacks remained largely consistent compared to an undefended model, indicating the attacks could still leverage an accurate or nearly accurate gradient to create an adversarial example. Unsurprisingly, gradient masking did not affect the performance of our black-box attack, as this attack operates independently of a model's gradient information.

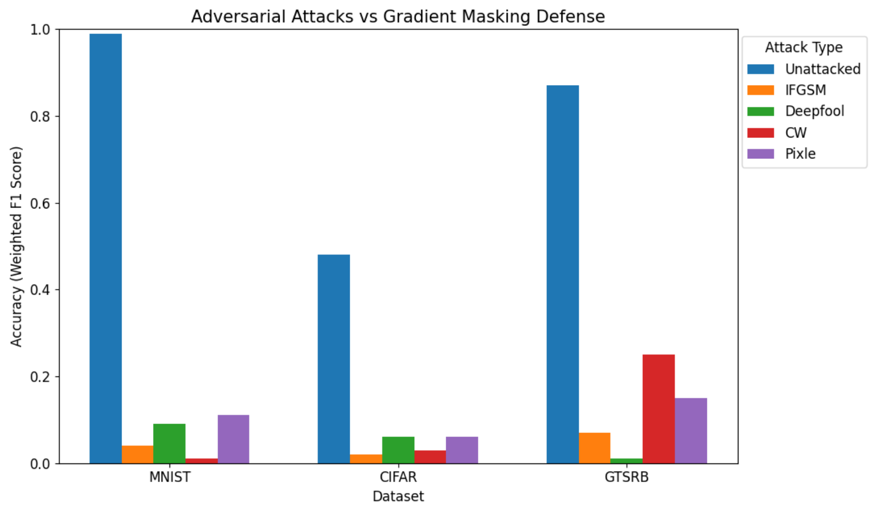


Figure 14: CNN accuracy after gradient masking.

The overall time to make a prediction did not meaningfully change after applying a gradient masking defense. This outcome aligns with expectations, given the operations we used in gradient masking were not particularly computationally expensive, resulting in a minimal impact on prediction time.

l

Prediction Time	MNIST	CIFAR	GTSRB
No Attack	0.002	0.002	0.03
IFGSM	0.11	0.06	0.22
Deepfool	0.36	0.24	0.77
CW	0.82	0.68	0.57
Pixel Swap	0.91	0.76	1.36

Table 13: Prediction time after gradient masking.

To summarize, our implementation of gradient masking proved ineffective as it both decreased performance on clean data and failed to improve performance on perturbed data. While gradient masking theoretically holds promise for enhancing the robustness of neural networks, our implementation ultimately did not work, highlighting the need for further refinement to realize its potential effectiveness.

3.5.4 Defensive Distillation

Defensive distillation emerged as one of our most effective defense methods, consistently increasing prediction accuracy across a variety of scenarios. Particularly noteworthy is its effectiveness against the deepfool attack, moving the classification accuracy from nearly to 0% to closely resembling the performance on clean data. Furthermore, defensive distillation exhibited strong performance against the pixel swap attack, nearly doubling the prediction accuracy compared to the pre-defense state. However, as expected, the CW attack posed a significant challenge to defensive distillation, showcasing the strategies susceptibility to attacks specifically engineered to beat it. This weakness was also present in the IFGSM attack, indicating defensive distillation has a significant vulnerability to that classification of attack. Interestingly, defensive distillation improved the prediction accuracy on clean data for the CIFAR and GTSRB models, which can likely be attributed to the benefits of the greater adaptability provided by training on soft labels.

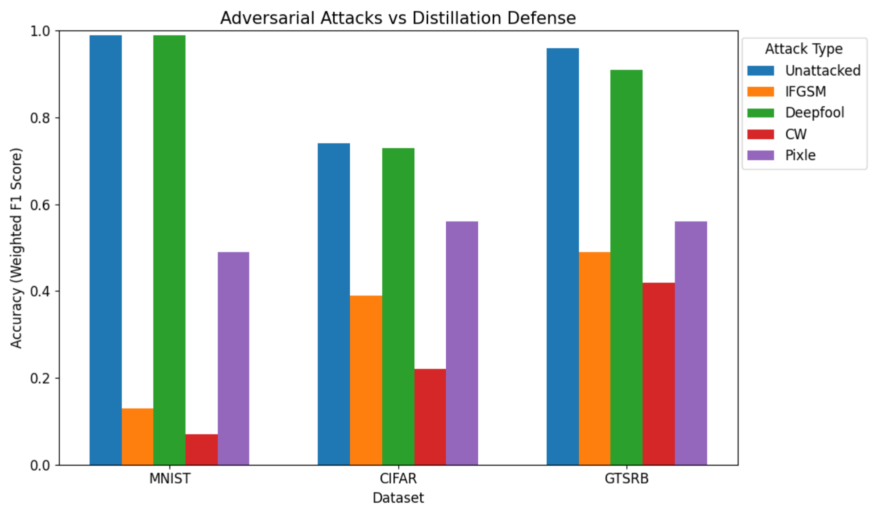


Figure 15: CNN accuracy after defensive distillation.

Once again, the prediction time remained largely unchanged compared to the original model, as the operations involved in a distilled model do not significantly differ from those in the base model. Instead, the time disparity occurs in the training phase as an additional model is required to be trained for defensive distillation.

l

Prediction Time	MNIST	CIFAR	GTSRB
No Attack	0.003	0.002	0.03
IFGSM	0.10	0.04	0.16
Deepfool	0.34	0.21	0.58
CW	0.62	0.67	0.71
Pixel Swap	0.77	0.77	1.40

Table 14: Prediction time after defensive distillation.

To conclude, defensive distillation proved to be one of our more effective defensive methods as it increased model performance on most attacks and enhanced accuracy on clean data. However, its widespread adoption as a defensive technique has led to the development of many attack strategies, such as the CW attack, designed to exploit its weaknesses. Therefore, while defensive distillation has proved to be an effective defense, its well known vulnerabilities necessitate alternate defense strategies to be adopted as well.

3.6 Final Evaluation

We conclude our examination of attacks and defenses by conducting a complete performance evaluation to assess whether any attacks or defenses consistently exhibited a dominant performance.

One of the key takeaways from our study is the dominant performance of our defenses against the deepfool algorithm compared to other attack methods. We hypothesize this phenomenon arises because the deepfool attack perturbs images so they lie very close to the prediction boundary. Consequently, only a minor defensive adjustment is necessary to correct the model's classification. A similar effect is observed with the pixel swap attacks; however, the more extreme perturbations associated with this attack reduces the prominence of this effect compared to a deepfool attack.

Our defenses generally worked better on the MNIST dataset than the CIFAR and GTSRB datasets. We attribute this to the lower complexity in the images of the MNIST dataset, which allowed for a more robust initial model, compared to the more intricate images of the other datasets where detecting noise can become more challenging.

Ultimately, adversarial example training emerged as the best performing defense overall, closely followed by defensive distillation and feature smoothing. While feature smoothing was very effective against deepfool and the pixel swap attack, it ultimately exhibited significantly lower capabilities against other attacks. Conversely, adversarial example training and defensive distillation were not extraordinarily successful against one type of attack, but their overall impact was more impressive. We conclude that training a model to recognize adversarial noise and features is the most effective technique to enhance the robustness of neural networks, as this training provides the broadest coverage across a variety of attack strategies.

4 Conclusion

In conclusion, our research highlighted the susceptibility of neural networks to adversarial attacks, as we designed four potent attack strategies, each exploiting distinct network vulnerabilities. While our defenses against these attacks exhibited promise overall, none proved universally effective against all types of attacks. This underscores the inherent challenge in designing effective defenses for neural networks, as all networks retain some level of vulnerability regardless of defense.

We also determined that our network-based defenses, defensive distillation and adversarial example training, generally outperformed image-based defenses like feature smoothing and gradient masking. This disparity likely emerges from the network-based defenses' ability to train models to discern perturbations, enabling adaptability to detect a diverse array of noise patterns. This contrasts with image-based defenses reliance on constraining the noise to a format recognizable by base models. Consequently, network-based models display greater resiliency to perturbed data, as image-based defenses will always fail if they are unable to effectively mitigate noise.

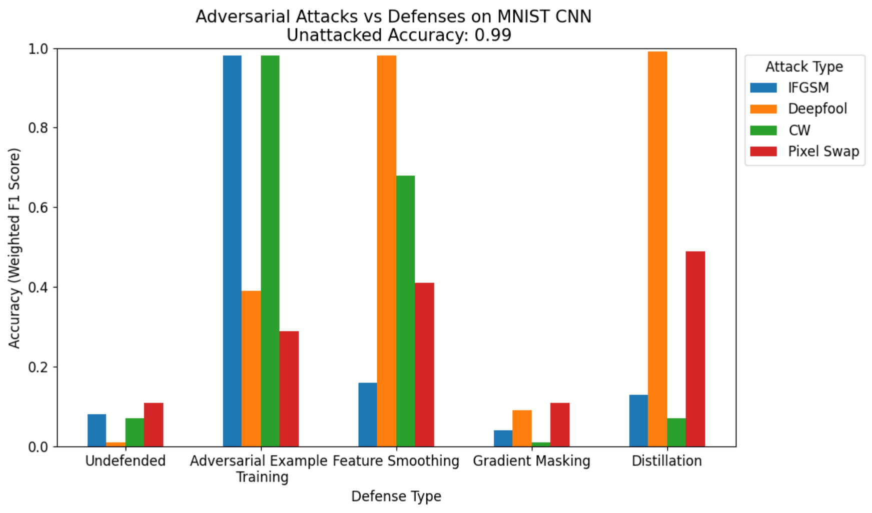


Figure 16: Summarizes performance of all defenses against all attacks on the MNIST dataset.

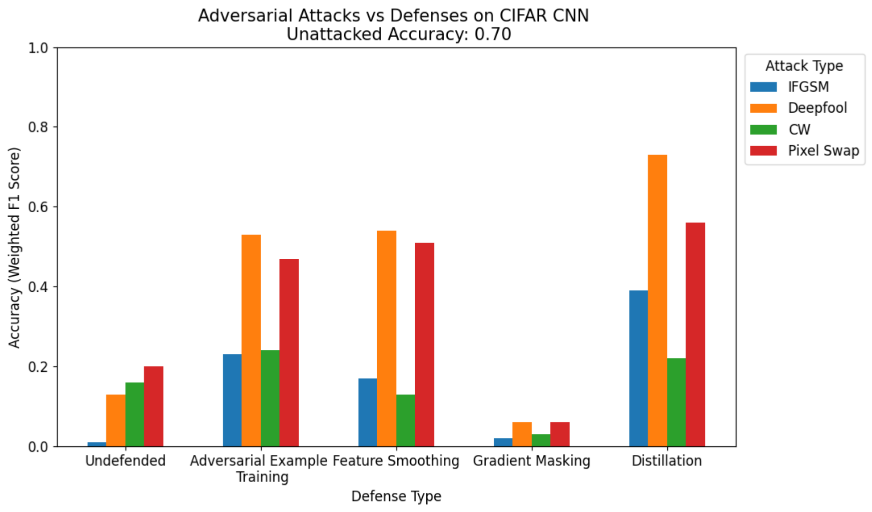


Figure 17: Summarizes performance of all defenses against all attacks on the CIFAR dataset.

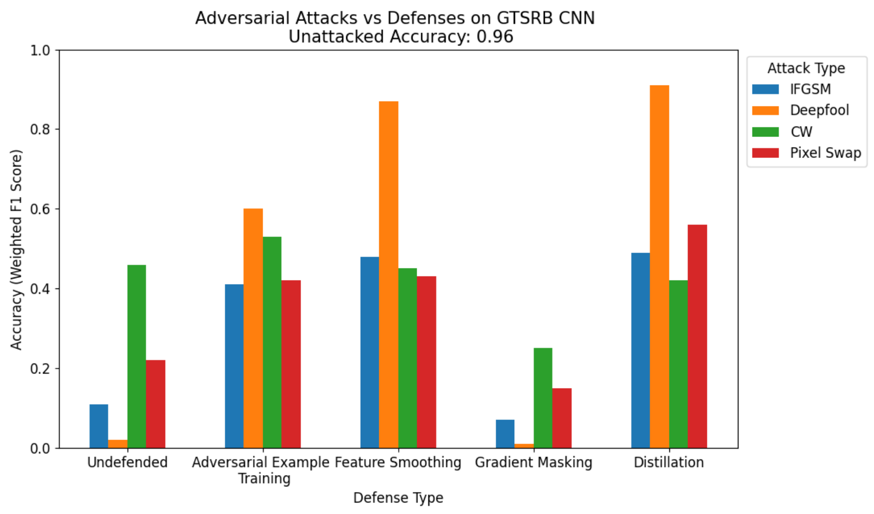


Figure 18: Summarizes performance of all defenses against all attacks on the GTSRB dataset.

4.1 Future Work

Although we concluded there are no perfect defenses against adversarial attack, we propose some additional approaches to enhance defensive performance beyond the development of alternate defense mechanisms. Gradient masking, in particular, represents an intriguing avenue of exploration, with the potential to greatly limit the impact of white-box attacks. There are other forms of implementing gradient masking, such as incorporating non-differentiable functions to a neural networks architecture, which addresses the linearity in neural networks that often render them susceptible to attack. Another prospective method involves deliberately introducing a vanishing or exploding gradient problem to render the gradient unusable for attacks, although this approach may compromise unattacked accuracy by impeding model training and prediction accuracy.

Another area of defense to explore is against targeted attacks. While we briefly experimented with designing attacks designed to converge upon a specified classification, further investigation into the impacts of defense strategies against these attacks is warranted. As targeted attacks are more powerful and applicable to the real world than untargeted attacks, it is important to measure our defenses against targeted attacks to truly evaluate their effectiveness.

Finally, our preliminary findings suggest that combining multiple defense methodologies can produce a superior model performance over using a singular defense. Given each defense has different strengths and weaknesses, integrating defenses with different components can help hide the weakness inherent to a particular approach. Notably, our experiments produced encouraging results when combining adversarial example training with feature smoothing, as the model capitalized on the strengths of each defense while mitigating their respective weaknesses. With further refinement and extensive testing, the integration of multiple adversarial defenses holds promise in providing robust protection against a variety of adversarial attacks, while preserving model performance on clean data.

References

[1] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, August 2017.

[2] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples, July 2018. arXiv:1802.00420 [cs].

[3] Nicholas Carlini and David Wagner. Towards Evaluating the Robustness of Neural Networks, March 2017. arXiv:1608.04644 [cs].

[4] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[5] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting Adversarial Attacks with Momentum, March 2018. arXiv:1710.06081 [cs, stat].

[6] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples, March 2015. arXiv:1412.6572 [cs, stat].

[7] Jonathan Helland and Nathan VanHoudnos. On the human-recognizability phenomenon of adversarially trained deep image classifiers, December 2020. arXiv:2101.05219 [cs].

[8] Chunming Jiang and Yilei Zhang. Adversarial Defense via Neural Oscillation inspired Gradient Masking, November 2022. arXiv:2211.02223 [cs].

[9] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images.

[10] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. *IEEE Transactions on Neural Networks and Learning Systems*, 33(12):6999–7019, December 2022.

[11] Hongshuo Liang, Erlu He, Yangyang Zhao, Zhe Jia, and Hao Li. Adversarial Attack and Defense: A Survey. *Electronics*, 11(8):1283, January 2022.

[12] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E. Alsaadi. A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26, April 2017.

[13] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. DeepFool: a simple and accurate method to fool deep neural networks, July 2016. arXiv:1511.04599 [cs].

[14] Keiron O'Shea and Ryan Nash. An Introduction to Convolutional Neural Networks, December 2015. arXiv:1511.08458 [cs].

[15] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597, May 2016. ISSN: 2375-1207.

[16] Jary Pomponi, Simone Scardapane, and Aurelio Uncini. Pixle: a fast and effective black-box attack based on rearranging pixels. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, July 2022. ISSN: 2161-4407.

[17] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 32:323–332, August 2012.

[18] Jiawei Su, Danilo Vasconcellos Vargas, and Sakurai Kouichi. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, October 2019. arXiv:1710.08864 [cs, stat].

[19] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, February 2014. arXiv:1312.6199 [cs].

[20] Weilin Xu, David Evans, and Yanjun Qi. Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks. In *Proceedings 2018 Network and Distributed System Security Symposium*, 2018. arXiv:1704.01155 [cs].

[21] Reda Yacouby and Dustin Axman. Probabilistic Extension of Precision, Recall, and F1 Score for More Thorough Evaluation of Classification Models. In Steffen Eger, Yang Gao, Maxime Peyrard, Wei Zhao, and Eduard Hovy, editors, *Proceedings of the First Workshop on Evaluation and Comparison of NLP Systems*, pages 79–91, Online, November 2020. Association for Computational Linguistics.