Project 2: Comparative Analysis of Adversarial Attacks and Defense Methods
Group 11

Machine Intelligence Course
Johns Hopkins University

Anita Madani smadani4@jh.edu
Namitha Guruprasad ngurupr1@jh.edu
Mahban Gholijafari mgholij1@jh.edu

Abstract

In this project, we explore and contrast the impacts of multiple adversarial attacks, including Projected Gradient Descent (PGD), noise-based attacks, Fast Gradient Sign Method (FGSM), Carlini & Wagner (C&W), DeepFool, Feature importance-aware Attack(FIA), and Jacobian-based Saliency Maps(JSM) on three distinct neural network architectures, using the CIFAR-10 dataset. The models evaluated include a custom-trained simple Convolutional Neural Network (CNN), a pre-trained ResNet-18, and a Tiny Vision Transformer (Tiny-VIT) which has been specifically trained on the CIFAR-10 dataset[1]. We employ a range of defense strategies, such as adversarial training, denoising through trained DnCNN, and regularization techniques, to mitigate the effects of these attacks. The performance of each model is meticulously assessed by comparing their accuracy under various adversarial conditions. This comparative analysis aims to elucidate the relative effectiveness and shortcomings of each attack and defense method, providing insight into their practical implications in securing neural networks against adversarial threats. All of our code and results are available on the GitHub repository linked below [1] and also on Google Drive [2].

1 Introduction

In this project, we're investigating several adversarial attacks on neural networks. We've chosen a mix of attack methods—Projected Gradient Descent (PGD), noise-based attacks, Fast Gradient Sign Method (FGSM), Carlini & Wagner (C&W), DeepFool, JSM, and FIA—each bringing a unique way to test the defenses of neural networks. PGD is noted for its effectiveness, though it requires a lot of computational resources. Noise-based attacks introduce random disturbances to assess robustness, but their lack of precision can be a drawback. FGSM is quick, using single-step updates along the gradient direction, yet this might sacrifice accuracy. C&W techniques are intricate, designed to produce minimal, stealthy perturbations, and DeepFool efficiently identifies the least amount of change needed to mislead classifiers, providing a refined approach. JSM focuses on exploiting the model's sensitivity to input features, using gradients to identify

[1] https://github.com/nam1410/JHU_EN.520.650.01.SP24_Machine_Intelligence.git

[2] https://drive.google.com/drive/folders/1ccUT3TrwcP9btn36y1bKxq8LlPH03gQQ?usp=drive_link

key components most affecting output. FIA, on the other hand, assesses the impact of each feature on the model's decision-making process, helping to pinpoint vulnerabilities in model behavior.

To counter these attacks, we employ defenses like adversarial training, where models learn from adversarial examples during training to enhance their resilience. We also utilize DnCNN, which we first train on noisy images, and use the trained model to clean images before they're processed by the network, helping maintain data integrity. Additionally, regularization techniques are applied to decrease the model's sensitivity to minor input variations, thereby minimizing the effectiveness of adversarial perturbations.

These attack and defense mechanisms are tested across three different neural network architectures: a custom-trained simple Convolutional Neural Network (CNN), a pre-trained ResNet-18, and a Tiny Vision Transformer (Tiny-VIT) specifically trained on the CIFAR-10 dataset. This variety allows us to evaluate how different architectures resist adversarial threats. Ultimately, we compare the performance of each model by assessing their accuracy under various adversarial conditions, utilizing the CIFAR-10 dataset as our experimental platform.

Through this analysis, we aim to identify which attack and defense combinations are most effective and explore the practical implications of securing neural networks against adversarial threats.

2 Adversarial attacks Overview

In the field of machine learning security, adversarial attacks represent a critical challenge, exploiting model vulnerabilities to induce errors. These attacks vary in their method of access to the model's information—ranging from white-box (complete access), semi-white-box (limited access), to black-box (no access)—and in their objectives, either targeted (specific misclassification) or non-targeted (any incorrect output).

Noise attacks are the simplest form of adversarial attacks, falling under the black-box category and typically non-targeted. They involve adding random noise to input images to disrupt model performance without requiring any knowledge about the model's parameters or architecture. Although simple, they are less effective than more sophisticated techniques. The Fast Gradient Sign Method (FGSM) is a white-box attack that utilizes the gradients of the neural network to create perturbations which maximize the model's loss, making it vulnerable either to targeted or non-targeted attacks. FGSM is efficient due to its single-step nature, but it might not explore the perturbation space thoroughly. Projected Gradient Descent (PGD) is considered one of the most potent adversarial techniques, also a white-box attack like FGSM, but it uses an iterative method, making it more effective at finding perturbations that mislead the model. It can be applied in both targeted and non-targeted scenarios, providing a deeper exploration of adversarial space compared to FGSM. The Feature Importance-aware Attack (FIA) represents a semi-white-box approach, where partial knowledge about the model, such as feature importance, is used to craft the attack. This makes FIA particularly effective in scenarios where some but not full internal knowledge of the model is available. Jacobian-based Saliency Map Attack (JSMA) is another white-box method, focusing on altering a few specific pixels that have the most significant impact on the output as determined by the model's Jacobian matrix. It is generally used for targeted attacks, aiming to achieve specific misclassification with minimal changes to the input image. DeepFool is an untargeted

attack method that iteratively perturbs an image until the model misclassifies it. Falling under the white-box category, it assumes full knowledge of the model's architecture and parameters and aims to find the minimal perturbations necessary for misclassification, providing insights into the model's vulnerability to small but effective alterations. Lastly, the Carlini & Wagner (C&W) attack, a white-box technique, is known for its effectiveness in bypassing defensive measures that are resistant to other attack methods. It can be configured for either targeted or non-targeted objectives, focusing on crafting perturbations that are harder to detect yet very effective at deceiving the models.

Comparing these methods, it is evident that the choice of attack depends on the attacker's access to model information and the specific objectives of the attack. Techniques like PGD and C&W offer high effectiveness but require deep knowledge of the model, making them suitable for scenarios where such information is obtainable. In contrast, methods like noise attacks provide a rudimentary approach applicable in more restricted settings where the model's inner workings are unknown. Each method has its trade-offs between complexity, effectiveness, and the level of knowledge required about the target model. Also, iterative methods like PGD, DeepFool, and C&W take more time due to their need to fine-tune the adversarial examples precisely. In contrast, single-step methods like FGSM offer a quicker, albeit potentially less refined, solution. Noise attacks are the quickest and simplest but at the cost of lower efficacy and predictability in their outcomes. Each method represents a trade-off between the time investment required and the level of threat posed to the target model.

Table 1: Classification of Adversarial Attack Methods

Attack Method	Attack Type	Targeted/Non-Targeted
Noise	Black-box	Non-targeted
FGSM	White-box	Non-targeted
PGD	White-box	Non-targeted
FIA	Semi-white-box	Targeted
JSMA	White-box	Targeted
DeepFool	White-box	Targeted
C&W	White-box	Targeted

2.1 Noise

In this part, we attempted to attack the model by adding random Gaussian noise to each image. The intensity of the noise is dictated by the 'epsilon' value, which represents the strength of our attack. This basic form of an adversarial attack is not targeted; it simply alters the image by adding noise. The visibility of changes in the perturbed image depends on the epsilon value; higher values result in stronger and more noticeable attacks. To implement this, we created the adversarial_noise function using the PyTorch library, generating noise with torch.randn_like(images) that mimics the image's shape, scales it by epsilon, and adds it to the original image to produce the perturbed version. At the end of the function, we use torch.clamp to clip each pixel's values between 0 to 255 to ensure the perturbed image remains valid.

2.2 Fast Gradient Sign Method (FGSM)

To implement the Fast Gradient Sign Method (FGSM) attack, we designed the FGSM function that requires three parameters: a neural network model, input images, and the epsilon value. This method introduces perturbations by utilizing the gradient of the loss function with respect to the inputs, intending to maximally increase the loss. This adjustment is based on the sign of the gradient and the epsilon value, which determines the strength of the attack.1

$$x_{\text{adv}} = x + \epsilon \cdot \text{sign}\left(\nabla_x \mathcal{L}(h(x, w), y)\right) \tag{1}$$

The process includes several steps: enabling gradient tracking for the images in PyTorch, obtaining the model's output, applying cross-entropy loss, and computing this loss with respect to the provided labels. After setting up the loss, we clear any existing gradients from previous operations and initiate backpropagation using the loss.backward() method. This calculates the gradients, which we then capture from image.grad.data. We isolate the sign of these gradients using the .sign() function. The original images are modified by adding the epsilon-scaled sign of the gradient, ensuring uniform perturbation intensity across the entire image, reliant solely on the epsilon value.

To finalize, we clamp the pixel values of the perturbed images to be between 0 and 255 using torch.clamp, ensuring the image remains within a valid pixel range. This method allows for quick execution of both targeted and non-targeted attacks, though it may lack fine-tuning in adversarial effectiveness due to the broader, less iterative steps used.[2]

2.3 Projected Gradient Descent (PGD)

Projected Gradient Descent (PGD) enhances the FGSM method by introducing iterative perturbations. After each perturbation step, PGD employs a projection function to ensure the adversarial examples remain within the epsilon-ball of the original image.2

$$x_{\text{adv}}^t = \Pi_\epsilon \left(x_{\text{adv}}^{t-1} + \alpha \cdot \text{sign}\left(\nabla_x \mathcal{L}(h(x_{\text{adv}}^{t-1}, w), y)\right)\right) \tag{2}$$

This projection is effectively implemented using the torch.clamp function, which restricts the adjusted pixel values to ensure they stay within the range of [-epsilon, epsilon] of the original image, maintaining the integrity of the perturbations and keeping them subtle. To implement the Projected Gradient Descent (PGD) attack, we adapt the approach used for FGSM by introducing iterative adjustments. The PGD function requires the neural network model, input images, and the epsilon value, with additional parameters for num_iter to control the number of iterations and alpha to set the step size, which dictates the strength of each attack iteration. During the PGD attack, we enable gradient tracking on the images, compute the model's output, apply cross-entropy loss, and calculate this loss relative to the labels. We clear existing gradients and perform backpropagation. The image is then updated by adding the epsilon-scaled gradient sign multiplied by alpha. To ensure that perturbations remain within acceptable bounds, we use the operation perturbed_image = torch.clamp(attacked_image - image, -epsilon, epsilon).

PGD's iterative approach allows for finer control compared to the single-step FGSM, making it potentially more effective by adapting and refining the perturbations with each iteration, thus providing a balance between attack potency and image validity.PGD is also regarded as the strongest first-order attack. [3]

2.4 Carlini & Wagner Attack (C&W)

Carlini and Wagner (C&W) [4] attack is one of the most powerful attacks, which uses three different vector norms: 1) the L_2 attack uses a smoothing of clipped gradient descent approach, displaying low distortion; 2) the L_0 attack uses an iterative algorithm that, at each iteration, fixes the pixels that do not have much effect on the classifier and finds the minimum amount of pixels that need to be altered; and 3) the L_∞ attack also uses an iterative algorithm with an associated penalty, penalizing every perturbation that exceeds a predefined value, formally defined as:

$$\min \left(c \cdot f(x+\delta) + \sum_i \left[(\delta_i - \tau)^+ \right] \right), \tag{3}$$

where δ is the perturbation, is the penalty threshold (initially 1, decreasing in each iteration), and c is a constant. The value for c starts as a very low value (e.g., 10^{-4}), and each time the attack fails, the value for c is doubled. If c exceeds a threshold (e.g., 10^{10}), it aborts the search[5].

2.5 Deep Fool

DeepFool [6] is an iterative attack that stops when the minimal perturbation that alters the model output is found, exploiting its decision boundaries. It finds the minimum perturbation for an input x_0, corresponding to the vector orthogonal to the hyperplane representing the decision boundary[5]. Formally, for a given classifier, we define an adversarial perturbation as the minimal perturbation r that is sufficient to change the estimated label $\hat{k}(x)$:

$$\Delta(x; \hat{k}) := \min_r \|r\|_2 \quad \text{subject to} \quad \hat{k}(x+r) \neq \hat{k}(x). \tag{4}$$

where x is an image and $\hat{k}(x)$ is the estimated label. As a multi-class classifier can be viewed as aggregation of binary classifiers, we implemented the binary classifiers. That is, we assume here $\hat{k}(x) = sign(f(x))$, where f is an arbitrary scalar-valued image classification function $f : R_n \to R$. Assuming now that f is a general binary differentiable classifier, we adopt an iterative procedure to estimate the robustness $\Delta(x_0; f)$. Specifically, at each iteration, f is linearized around the current point x_i and the minimal perturbation of the linearized classifier is computed as

$$\arg\min_{r_i} \|r_i\|_2 \quad \text{subject to} \quad f(x_i) + \nabla f(x_i)^T r_i = 0. \tag{5}$$

The perturbation r_i at iteration i of the algorithm is computed using the closed form solution in Eq. 6, and the next iterate x_{i+1} is updated.

$$r_*(x_0) := \arg\min \|r\|_2 \quad \text{subject to} \quad \text{sign}(f(x_0+r)) \neq \text{sign}(f(x_0)) = -\frac{f(x_0)}{\|w\|_2} w. \tag{6}$$

The algorithm stops when x_{i+1} changes sign of the classifier.

2.6 Jacobian-based Saliency Maps (JSM)

Jacobian-based Saliency Maps (JSM) uses forward derivatives to compute model gradients, which offers an alternative to gradient descent method. This technique identifies regions in the input that are susceptible to produce adversarial examples. After that, JSM generates saliency maps that highlight the features that need to be altered by an adversary. Finally, to prove the effectiveness of JSM, only those adversarial examples that were correctly identified by humans were used to fool neural networks[5]. Adversarial saliency maps indicate which input features an adversary should perturb in order to effect the desired changes in network output most efficiently, and are thus versatile tools that allow adversaries to generate broad classes of adversarial samples. The saliency map is based on the forward derivative, as this gives the adversary the information needed to cause the neural network to misclassify a given sample. More precisely, the adversary wants to misclassify a sample X such that it is assigned a target class $t \neq label(X)$. To do so, the probability of target class t given by F, $F_t(X)$, must be increased while the probabilities $F_j(X)$ of all other classes $j \neq t$ decrease, until $t = argmax_j F_j(X)$. The adversary can accomplish this by increasing input features using the following saliency map $S(X_t)$ where i is an input feature.

$$S^+(x_i, c) = \begin{cases} 0 & \text{if } \frac{\partial f(x)_c}{\partial x_i} < 0 \text{ or } \sum_{c' \neq c} \frac{\partial f(x)_{c'}}{\partial x_i} > 0, \\ -\frac{\partial f(x)_c}{\partial x_i} \cdot \sum_{c' \neq c} \frac{\partial f(x)_{c'}}{\partial x_i} & \text{otherwise.} \end{cases} \tag{7}$$

The condition specified on the first line rejects input components with a negative target derivative or an overall positive derivative on other classes. The product on the second line allows us to consider all other forward derivative components together in such a way that we can easily compare $S(X_t)[i]$ for all input features. In summary, high values of $S(X_t)[i]$ correspond to input features that will either increase the target class, or decrease other classes significantly, or both. By increasing these features, the adversary eventually misclassifies the sample into the target class.[7]

2.7 Feature Importance-aware Attack (FIA)

Feature Importance-aware Attack (FIA) considers the object aware features that dominate the model decisions, using the aggregate gradient (gradients average concerning the feature maps). This approach avoids local optimum, represents transferable feature importance, and uses the aggregate gradient to assign weights identifying the essential features. Furthermore, FIA generates highly transferable adversarial examples when extracting the feature importance from multiple classification models.[5] For simplicity, let f denote the source model, and the feature maps from the k-th layer are expressed as $fk(x)$. Since feature importance is proportional to how the features contribute to the final decision, an intuitive strategy is to obtain the gradient w.r.t. $f_k(x)$ as written in the following.

$$\Delta_k^x = \frac{\partial \mathcal{L}(x, t)}{\partial f_k(x)}, \tag{8}$$

where $l(\cdot, \cdot)$ denotes the logit output with respect to the true label t. However, the raw gradient Δ_k^x would carry model specific information. The raw gradient maps and raw feature maps are both visually noisy,i.e., pulses and large gradient on non-object regions,which may be caused by the model-specific solution space. To suppress model-specific information, they propose the aggregate gradient, which aggregates gradients from randomly

transformed x. The transformation is supposed to distort image details but preserve the spatial structure and general texture. Since semantically object-aware/important features/gradients are robust to such transformation but model-specific ones are vulnerable to the transforms, those robust/transferable features/gradients will be highlighted after aggregation, while the others would be neutralized. In this project, we adopt random pixel dropping (i.e., randomly mask) with the probability p_d. Therefore, the aggregate gradient can be expressed in the following.

$$\hat{\Delta}_k^x = \frac{1}{C} \sum_{n=1}^{N} \Delta_k^{x \circ M_n^{p_d}}, \quad M_{p_d} \sim \text{Bernoulli}(1 - p_d), \tag{9}$$

where the M_{p_d} is a binary matrix with the same size as x, and $\odot$ denotes the element-wise product. The normalizer C is obtained by $l_2 - norm$ on the corresponding summation term. The ensemble number N indicates the number of random masks applied to the input x. The aggregate gradient $\bar{\Delta}_k^x$ highlights the regions of robust and critical object-aware features that can guide the adversarial example towards the more transferable direction[8].

3 Defense Methods Overview

Despite the remarkable accomplishments in training neural networks, it is clear that even the most skilled models have their limitations. Several studies have shown the vulnerability of these models with the smallest variation in input data structure. This vulnerability to adversarial examples was first highlighted in the joint research paper by Google and New York University, "Intriguing properties of neural networks, 2014". There are now competitions to address this issue. In this project, we tested three different approaches for defense. [9]

3.1 Adversarial Training with Noise

To make a neural network model robust to these adversarial examples [10] proposed perturbation of inputs. The perturbation creates diversity in inputs by adding the original input to the sign of the gradient (Calculated relative to the training input) multiplied by a constant value. The creation of a perturbed image is given as:

$$\hat{x} = x + \epsilon sign(\Delta_x(J)) \tag{10}$$

where for x is the training input, $\hat{x}$ is a new perturbed image, $\Delta_x(J)$ is the gradient of the loss-function (J) with respect to the training input x, ϵ is a predetermined constant value- small enough to be discarded by a data storage apparatus due to limited precision of it, and the sign function outputs 1 if the input is positive, -1 if negative and 0 if zero. Training the network with perturbed input (adding noise to the inputs) extends the input distribution with its diversity. This makes the network robust against adversarial attacks. However, noise with zero mean and zero covariance is very inefficient at preventing adversarial examples. The expected dot product between any reference vector and such a noise vector is zero. This means that in many cases the noise will have essentially no effect rather than yielding a more difficult input[10]. In this project, we added Gaussian noise to all the input data during training.

3.2 Adversarial Training with L_2 Regularization

Our method is closely related to the adversarial training proposed by Goodfellow et al[10]. We therefore formulate adversarial training before introducing our method. The loss function of adversarial training in[10] can be written as

$$L_{\text{adv}}(x_1, \theta) := D[q(y|x_1), p(y|x_1 + r_{\text{adv}}, \theta)] \tag{1}$$

where r_{adv} is defined as:

$$r_{\text{adv}} := \arg \max_{r;\|r\| \leq \epsilon} D[q(y|x_1), p(y|x_1 + r, \theta)] \tag{2}$$

where $D[p, p']$ is a non-negative function that measures the divergence between two distributions p and p'. The function $q(y|x_l)$ is the true distribution of the output label, which is unknown. The goal with this loss function is to approximate the true distribution $q(y|x_l)$ by a parametric model $p(y|x_l)$ that is robust against adversarial attack to x. Generally, we cannot obtain a closed form for the exact adversarial perturbation r_{adv}. However, we can approximate r_{adv} with a linear approximation of D with respect to r in Eq. 1 When the norm is L_2, adversarial perturbation can be approximated by Eq. 11.

$$r_{\text{adv}} \approx \frac{\epsilon\, g}{\|g\|_2}, \quad \text{where } g = \nabla_{x_1} D\left[h(y; y_l), p(y|x_1, \theta)\right] \tag{11}$$

Note that, for NNs, the gradient $\Delta_{x_l} D[h(y; y_l) p(y|x_l)]$ can be efficiently computed by backpropagation. By optimizing the loss function of the adversarial training in Eq. 1 based on the adversarial perturbation defined by Eq. 11, [10] was able to train a model with better generalization performance than the model trained with random perturbations[11].

3.3 Denoising (DnCNN)

In our approach to defending against adversarial attacks, we employ a denoising convolutional neural network (CNN) named DnCNN. This model is specifically designed to preprocess noisy images that have been compromised by adversarial attacks. As one of the simpler adversarial defense mechanisms, its primary function is to clean up the input data before it's processed further by other models.

The DnCNN operates by taking a noisy image as its input and processing it through its neural network architecture. This architecture begins with an input layer that receives the noisy image and uses a convolutional layer to transform the RGB input into a deeper feature space, typically starting with 64 feature maps. This initial transformation is crucial for preparing the image for more detailed processing. Following the input layer, the architecture features multiple convolutional layers, each followed by batch normalization and ReLU activation functions. The convolutional layers use kernels of size 3x3 with a padding of 1 to maintain the spatial dimensions of the feature maps throughout the network, ensuring that the edge information is not lost. Batch normalization helps stabilize the learning process by normalizing the outputs of the previous layers, speeding up convergence and improving the overall training dynamics. ReLU activation introduces non-linearity into the model, which is essential for learning complex patterns in the data. The core of the network consists of a repetitive pattern of these layers, extended over

17 layers in total, where the bulk of noise reduction tasks are performed. The middle layers all maintain the same number of features (64), allowing the network to develop a deep representation of the noise characteristics without altering the underlying spatial dimensions. The final layer in the network is another convolutional layer that maps the deep feature representation back to the original three channels of the input image. This layer is crucial as it reconstructs the denoised image from the processed features. The output of this final layer is actually the noise that the network predicts is present in the input image.

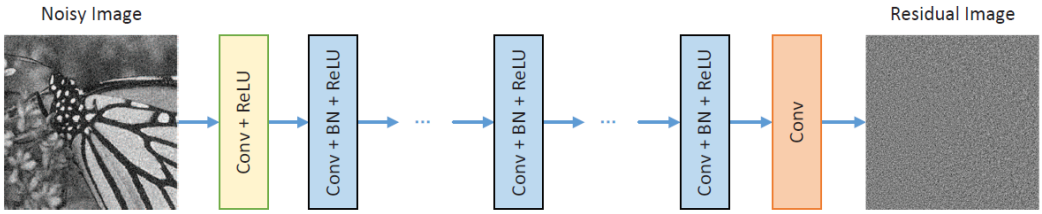

Figure 1: An illustration of the DnCNN architecture

The operation of the model culminates in subtracting this predicted noise from the original noisy image. This subtraction results in the final, clean image output, effectively reversing the noise effects. By the end of our training, the model achieved a notably low loss of 0.13, indicating effective learning and noise reduction capabilities. This reduction in noise directly contributed to a 10% improvement in the accuracy of the model on noise-affected images.

Additionally, we explored alternative denoising methods such as using OpenCV's fastN1Mean for colored image denoising. However, this approach did not yield an improvement in the accuracy against adversarial attacks, suggesting that our custom-trained model is more suited for this specific type of noise encountered in adversarial contexts.[12]

4 Results and Comparison

4.0.1 Dataset

We utilized the CIFAR-10 dataset [1], as mentioned in the Project description, comprising 60,000 color images. These images are 32x32 pixels in size and are categorized into 10 classes, with each class containing 6,000 images. The dataset is divided into a training set of 50,000 images and a test set of 10,000 images. Each image belongs to one of the following categories: plane, car, bird, cat, deer, dog, frog, horse, ship, or truck. CIFAR-10 is widely utilized for evaluating machine learning algorithms, particularly in image classification tasks, due to its diverse object categories and relatively small image dimensions, which present challenges for model generalization

4.0.2 Technical Details

Most of our coding is in PyTorch [13], developed by Facebook's AI Research lab, which offers a dynamic computation graph framework ideal for building and training deep learning models. It features automatic differentiation, GPU acceleration, and support for various neural network architectures, facilitating rapid prototyping and deployment. We used majorly Google Colab [14] that provides free access to GPU resources, typically T4,

with integration to our Google Drive. This offered us a collaborative environment for developing and sharing code.

4.1 Quantitative analysis

The following tables present a comparative analysis of various DCNN models' accuracies on the CIFAR-10 dataset both pre- and post-application of diverse adversarial attacks. These models underwent further training with the above three distinct defense mechanisms, namely adversarial training with noise, L2 regularization, and denoising, to assess the impact on their accuracies.

In our comparative study of models trained on the CIFAR-10 dataset, the Tiny ViT model stood out, achieving the highest accuracy at 78%. This was followed by our custom-trained CNN model, which reached 66% accuracy, and the fine-tuned ResNet-18, which varied between 42-46% accuracy.

The ResNet-18 model demonstrated strong resilience against the DeepFool attack, which had the least impact, while the PGD attack significantly reduced its accuracy, proving to be the most effective in compromising model performance. The CNN model showed considerable vulnerability to noise, which drastically affected its accuracy. Similarly, the DeepFool attack had minimal impact on this model. Due to limited computational resources, the ViT model was only tested against three attack types. Among these, the PGD attack reduced its accuracy the most, whereas FGSM had the least effect.

Adversarial training techniques, including noise incorporation and FGSM, were applied to the CNN model, enhancing its accuracy by approximately 1.76%. Training with noise was particularly effective, improving resistance to attacks more than training with FGSM, despite a 4% decrease in raw accuracy. L2 regularization on the pre-trained ResNet-18 slightly improved its baseline accuracy by about 2%, but it had minimal impact on performance under attack, enhancing it by less than 1%. Training the CNN model on noisy images significantly increased its resilience, boosting accuracy by about 10% under noisy conditions and 1-2% against other attacks such as FGSM, PGD, and DeepFool, though it was less effective against FIA and C&W attacks. Adversarial training with noise on the ViT model reduced its accuracy by roughly 50% but strengthened its defense against noise attacks, albeit without improving performance against other attack types like PGD and FGSM.

Regarding implementation times, L2 regularization proved to be the fastest method, followed by the denoising algorithm, with adversarial training taking the most time. Limited computational resources restricted the extent of adversarial training to only 10-15 epochs, which suggests that with enhanced resources, better results could be achievable.

Table 2: Best Defense Methods for Various Attack Types

Attack Type	**Best Defense Method**
FGSM (Fast Gradient Sign Method)	Adversarial Training with FGSM/ NOISE
PGD (Projected Gradient Descent)	Adversarial Training with FGSM/NOISE
Noise	DnCNN Denoising
FIA (Feature Adversaries)	Adversarial Training with FGSM/NOISE
JSM (Jacobian-based Saliency Map Attack)	Adversarial Training with NOISE
DeepFool	Adversarial Training with NOISE
C&W (Carlini & Wagner Attack)	Adversarial Training with NOISE

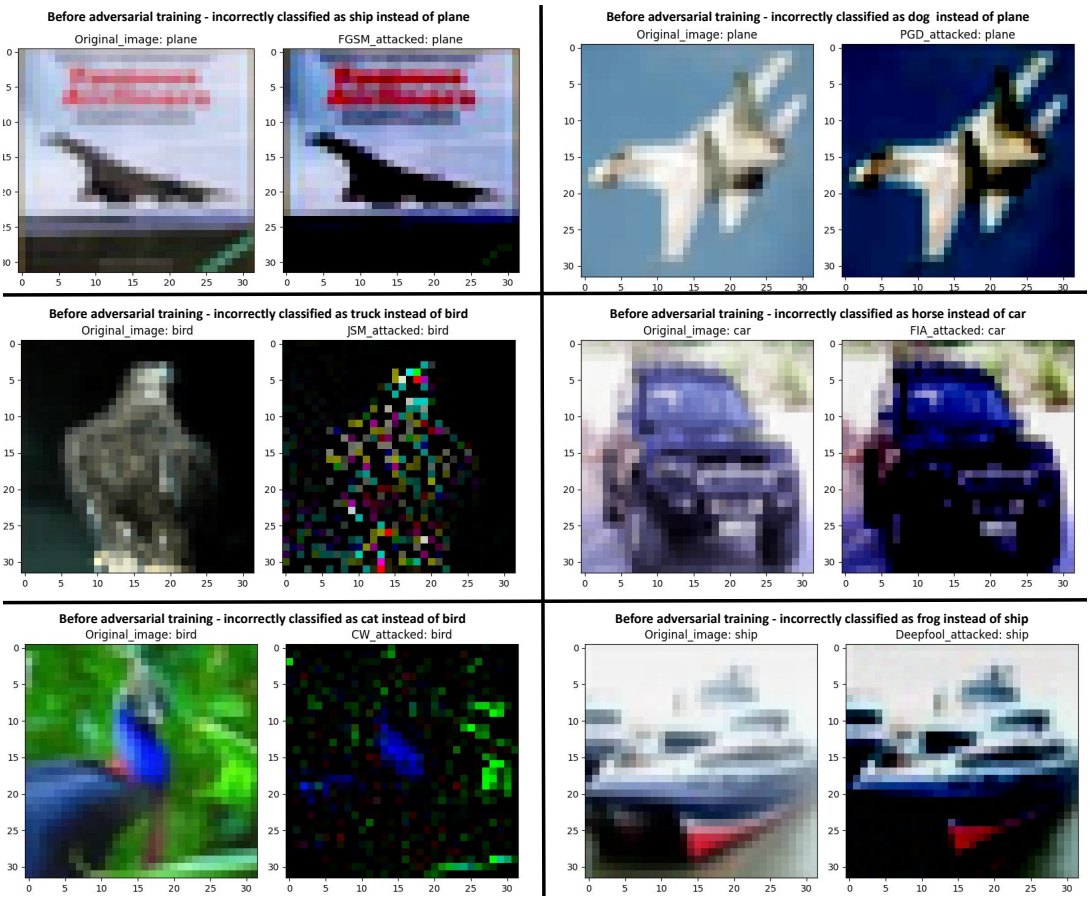

Figure 2: Impact of CNN-CIFAR10-Noise in Images Attacked with Adversaries

Table 3: Comparison of accuracy with adversarial attacks before and after training with Defense - Adversarial Training with Noise

Accuracy / Attack	CNN-CIFAR10 (%)	CNN-CIFAR10 after attack%	CNN-CIFAR10-Noise	CNN-CIFAR10-Noise after attack%
Noise	66.26	9.92	61.05	10.53
FGSM	66.26	38.23	61.05	40.13
PGD	66.26	38.25	61.05	40.16
C&W	66.26	11.33	61.05	13.17
Deep Fool	66.26	40.69	61.05	42.70
JSM	66.26	39.89	61.05	41.37
FIA	66.26	39.27	61.05	40.71
Difference (before and after adversarial training)				1.76

Note: CNN-CIFAR10 is a Convolutional based Network trained on CIFAR-10 dataset for 100 epochs. CNN-CIFAR10-Noise is CNN-CIFAR10 trained with adversarial Gaussian noisy inputs for 100 epochs.

Table 4: Comparison of model accuracy with adversarial attacks and training with FGSM

Accuracy / Attack	CNN-CIFAR10 (%)	CNN-CIFAR10 after attack%	CNN-CIFAR10-FGSM	CNN-CIFAR10-FGSM after attack%
Noise	66.26	9.92	62.02	10.23
FGSM	66.26	38.23	62.02	39.56
PGD	66.26	38.25	62.02	39.59
C&W	66.26	11.33	62.02	12.34
Deep Fool	66.26	40.69	62.02	41.97
JSM	66.26	39.89	62.02	40.93
FIA	66.26	39.27	62.02	40.29
Difference (before and after adversarial training)				1.17

Note: CNN-CIFAR10 is a Convolutional based Network trained on CIFAR-10 dataset for 100 epochs. CNN-CIFAR10-FGSM is CNN-CIFAR10 trained with adversarial FGSM noisy inputs for 100 epochs.

Table 5: Comparison of accuracy with adversarial attacks before and after training with Defense - L_2 regularization

Accuracy / Attack	ResNet18-CIFAR10 (%)	ResNet18-CIFAR10 after attack%	ResNet18-CIFAR10-L_2%	ResNet18-CIFAR10-L_2 after attack%
Noise	42.22	9.75	44.37	10
FGSM	42.22	9.17	44.37	10.01
PGD	42.22	9.36	44.37	10.01
C&W	42.22	14.03	44.37	12.99
Deep Fool	42.22	30.66	44.37	29.74
JSM	42.22	19.36	44.37	20.13
FIA	42.22	9.54	44.37	9.68
Difference (before and after regularization)				0.06

Note: ResNet18-CIFAR10 is ResNet18 trained on CIFAR-10 dataset for 10 epochs. The average increase in the accuracy across all attacks is 0.06%.

Table 6: Comparison of model accuracy with adversarial attacks and denoising defense method (DnCNN)

Accuracy / Attack	ResNet18-CIFAR10 (%)	ResNet18-CIFAR10 after attack%	ResNet18-CIFAR10-DnCNN after attack%	
Noise	41.46	17.19	26.58	
FGSM	41.46	12.44	15.58	
PGD	41.46	9.08	9.72	
C&W	41.46	22.3	21.27	
Deep Fool	41.46	19.81	20.96	
JSM	41.46	17.34	18.4	
FIA	41.46	21.62	20.54	
Average accuracy				19.00
Difference (before and after adversarial training)				1.89

Note: The model is ResNet18 trained on CIFAR-10 dataset for 10 epochs. As demonstrated in the table, the dnCNN model significantly enhances the accuracy of the noise-attacked model and also improves resilience against other attack methods by around 1-2 percent.

Table 7: Comparison of model accuracy with adversarial attacks on Tiny ViT

Accuracy / Attack	ViT-CIFAR10 (%)	ViT-CIFAR10 after attack%
Noise	78.6	12.58
FGSM	78.6	17.75
PGD	78.6	12.41

Note: The model is Tiny ViT trained on CIFAR-10 dataset for 200 epochs.

Table 8: Comparative Analysis of Model Accuracy Using Adversarial Noise Training

Accuracy / Attack	ViT-CIFAR10 (%)	ViT-CIFAR10 after attack%	ViT-CIFAR10-Noise%	ViT-CIFAR10-Noise after attack%
Noise	78.6	12.58	21.56	16.61
FGSM	78.6	17.75	21.56	13.96
PGD	78.6	12.41	21.56	10.23

Note: The model is Tiny ViT trained on CIFAR-10 dataset for 200 epochs.

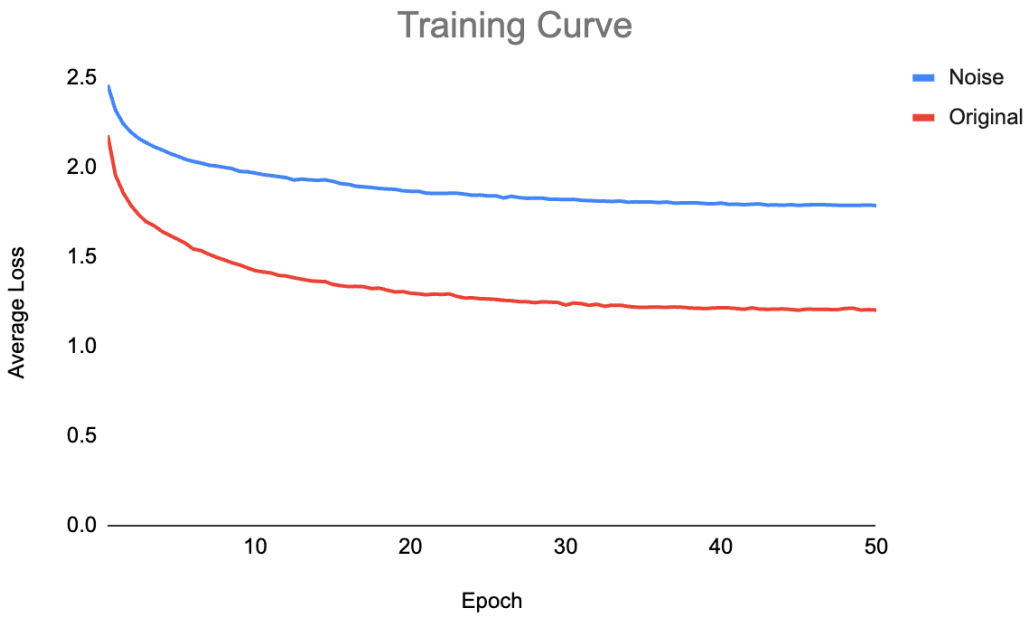

Figure 3: Training curves for CNN-CIFAR10 model before and after adversarial training

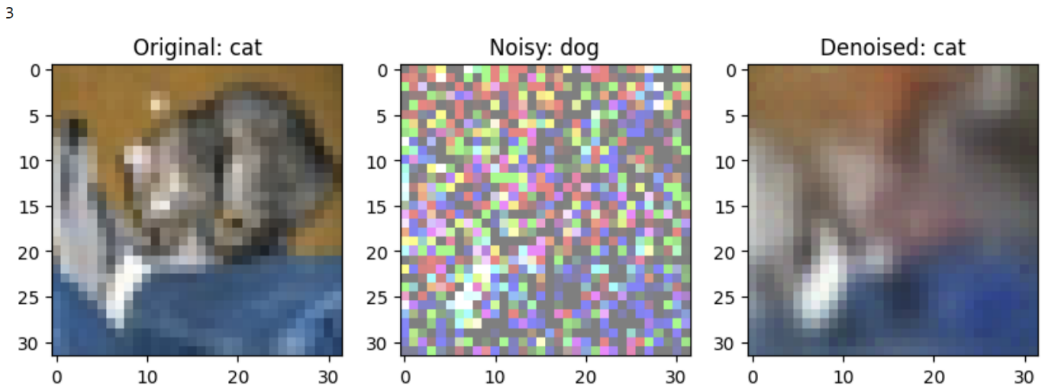

Figure 4: Impact of DnCNN on Images Attacked with Noise

4.2 A closer look at CNN-CIFAR10

An in-depth exploration of specific examples, particularly those involving the CNN-CIFAR10 model, gives insights into the impact of adversarial attacks on model behavior.

Notably, post-attack analysis reveals a tendency for the model to misclassify images, often assigning them to categories that bear little resemblance to their true content (for example, a plane misclassified as a dog). This highlights the disruptive influence of perturbations on the model's decision-making process and influences the severity of accuracy degradation. However, adversarial training incorporates noise-based defenses and we observe a reversal in the model's performance. In contrast to its earlier struggles, the CNN-CIFAR10-Noise model now exhibits a slight improvement in its classification accuracy (parallels with the values in **Table** 3), rectifying previous misclassifications and restoring confidence in its decision-making capabilities. This positive outcome, while small, is of significance, representing a step towards improving the model robustness in the face of adversarial challenges. Slight improvements were observed following adversarial training with noise and denoising, resulting in an approximate 2% increase in accuracy. These improvements were achieved despite the models being trained for a limited number of epochs (50). For instance, examining the training curves of the CNN-CIFAR10 model before and after adversarial training (as shown in **Fig.** 3) reveals promising indications of convergence over a larger number of training epochs.

5 Discussion and Conclusion

The preceding analysis reveals insights into the performance of different architectures when coupled with diverse denoising strategies. Tiny ViT emerges as a frontrunner, boasting a superior baseline accuracy of 78%. Adversarial training with noise visibly bolsters the robustness of CNN models, particularly against noise attacks, albeit with a slight compromise in raw accuracy.

The Adversarial attacks in our study play pivotal roles in evaluating and fortifying the robustness of machine learning models against adversarial attacks. While Gaussian noise serves as a foundational technique for introducing random perturbations, FGSM stands out for its simplicity and effectiveness in crafting adversarial examples. JSM and PGD leverage gradient information to generate perturbations and exhibit model vulnerabilities. CW and DeepFool, on the other hand, exhibit greater complexity but are potent adversaries capable of competing robust defenses. The implementation of denoising techniques and adversarial training in our project demands heightened computational resources.

Throughout our study, the issue of computational constraints emerged as a significant challenge, imposing limitations on the depth and breadth of our analyses. Despite our concerted efforts, these constraints hindered our ability to conduct comprehensive model training sessions and implement robust cross-validation methodologies such as early stopping. Additionally, the limited computational resources constrained our capacity to employ advanced hyperparameter optimization techniques like Optuna [15], which could have facilitated fine-tuning of model parameters for enhanced performance against adversarial attacks.

With access to additional computational resources, we would have extended model training durations, allowing for more exhaustive exploration of model architectures and optimization strategies. Moreover, we would have implemented stringent cross-validation protocols to ensure the generalizability of our findings across different datasets and experimental conditions. Furthermore, time and resource constraints precluded us from investigating the impact of additional loss terms such as Kullback-Leibler (KL) divergence alongside the cross-entropy classification loss. This exploration could have provided valu-

able insights into the quality of model predictions in the presence of adversarial attacks. Despite these limitations, our study yields promising results, underscoring the potential for future advancements in adversarial defense strategies with increased GPU resources.

6 Acknowledgement

We want to express our sincere gratitude to Dr. Rama Chellappa (instructor) and Daniel Olshvang and Michael Sprintson (teaching assistants) for their invaluable guidance and support throughout this research project with technical and conceptual aspects.

References

[1] A. Krizhevsky, "Learning multiple layers of features from tiny images," tech. rep., University of Toronto, 2009. Accessed: [insert date here].

[2] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *International Conference on Learning Representations (ICLR)* , 2015.

[3] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083* , 2017.

[4] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 ieee symposium on security and privacy (sp)*, pp. 39–57, Ieee, 2017.

[5] J. C. Costa, T. Roxo, H. Proença, and P. R. Inácio, "How deep learning sees the world: A survey on adversarial attacks & defenses," *IEEE Access* , 2024.

[6] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2574–2582, 2016.

[7] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *2016 IEEE European symposium on security and privacy (EuroS&P)*, pp. 372–387, IEEE, 2016.

[8] Z. Wang, H. Guo, Z. Zhang, W. Liu, Z. Qin, and K. Ren, "Feature importance-aware transferable adversarial attacks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 7639–7648, October 2021.

[9] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *International Conference on Learning Representations (ICLR)*, 2014.

[10] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572* , 2014.

[11] T. Miyato, S.-i. Maeda, M. Koyama, and S. Ishii, "Virtual adversarial training: a regularization method for supervised and semi-supervised learning," *IEEE transactions on pattern analysis and machine intelligence* **41**(8), pp. 1979–1993, 2018.

[12] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising," *IEEE Transactions on Image Processing* **26**(7), pp. 3142–3155, 2017.

[13] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," 2019.

[14] "Google colaboratory." `https://colab.research.google.com/`.

[15] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* , 2019.