CS685/785 Group Project

Adversarial Attack on a CNN Trained on CIFAR-10 dataset
FGSM + PGD White Box Attacks: An Analysis of the Attack Success Rates

Group Information

Group Name: Neurons

Group Members:

- Aayush Kushwaha
- Balaka Biswas
- Harsh Deshmukh
- Sahil Khandu Thorat

1 Objective

This project studies how a convolutional neural network (CNN) trained on the CIFAR-10 dataset can be intentionally misled by small, carefully constructed perturbations to the input images. The chosen project direction is adversarial attack, using two gradient based methods: the Fast Gradient Sign Method (FGSM) and Projected Gradient Descent (PGD). The main objective is to quantify how the classifier's accuracy and the attack success rate change under different perturbation budgets and, for PGD, different numbers of attack steps. A further goal is to compare FGSM and PGD in terms of their ability to reduce accuracy while keeping the perturbed images visually close to the original samples, and to illustrate these effects through visual examples and perturbation heatmaps. :contentReference[oaicite:0]index=0

2 Methodology

The methodology has two stages. First, a baseline CNN classifier is trained on the CIFAR-10 dataset to obtain a model with reasonable accuracy on clean test images. Second, the trained classifier is held fixed and used to generate adversarial examples with FGSM and PGD. Both attacks operate in a white box setting, where gradients of the loss with respect to the input image are available, and both constrain the perturbation using an L_∞ bound of size ϵ.

2.1 Baseline CNN and loss function

Let $x \in \mathbb{R}^{3\times32\times32}$ denote a CIFAR-10 image and $y \in \{0, \ldots, 9\}$ its class label. Before training, each image is normalized channelwise using the dataset mean μ_c and standard deviation σ_c:

$$\tilde{x}_c = \frac{x_c - \mu_c}{\sigma_c}, \quad c \in \{1, 2, 3\}.$$

The classifier f_θ is a CNN with three convolutional blocks followed by two fully connected layers. Each block consists of a 3×3 convolution, a ReLU activation, and 2×2 max pooling, which progressively reduces the spatial resolution from 32×32 to 16×16, then 8×8, and finally 4×4. The resulting feature map is flattened and passed through a fully connected layer with 256 hidden units and ReLU, followed by a final linear layer that outputs logits $z = f_\theta(\tilde{x}) \in \mathbb{R}^{10}$.

The model is trained using the standard multi class cross entropy loss

$$\mathcal{L}(\tilde{x}, y; \theta) = -\log\left(\frac{\exp(z_y)}{\sum_{k=1}^{10}\exp(z_k)}\right),$$

where $z = f_\theta(\tilde{x})$. During adversarial example generation, the model parameters θ are kept fixed, and gradients of $\mathcal{L}$ are computed with respect to the input image $\tilde{x}$.

2.2 Fast Gradient Sign Method (FGSM)

FGSM constructs a single step perturbation in the direction of the sign of the gradient of the loss with respect to the input. For an input $\tilde{x}$ and true label y, the attack computes the gradient

$$g = \nabla_{\tilde{x}} \mathcal{L}(\tilde{x}, y; \theta),$$

and defines the adversarial example

$$\tilde{x}_{\text{adv}} = \tilde{x} + \epsilon \cdot \text{sign}(g),$$

where $\epsilon > 0$ is the perturbation budget and $\text{sign}(\cdot)$ is applied elementwise. This corresponds to an untargeted attack that increases the loss for the true label while respecting an L_∞ constraint $\|\tilde{x}_{\text{adv}} - \tilde{x}\|_\infty \leq \epsilon$.

In practice, the perturbed image is also clipped to stay within a valid normalized range $[\tilde{x}_{\min}, \tilde{x}_{\max}]$, which corresponds to the range induced by the original CIFAR-10 normalization. The procedure for a batch can be summarized as follows.

Pseudocode: FGSM attack

```
Input: model f, loss L, batch of images X, labels Y, epsilon
Output: batch of adversarial images X_adv

1. X ← detach(X); enable gradient for X
2. Z ← f(X)                          # forward pass
3.  ← L(Z, Y)                        # cross entropy loss
4. Compute gradient G ←  / X         # backpropagation
5.  ← epsilon · sign(G)              # elementwise sign
6. X_adv ← X +
7. X_adv ← clip(X_adv, X_min, X_max)
8. Return X_adv (detached from graph)
```

Here $X_{\min}$ and $X_{\max}$ denote the lower and upper bounds for normalized pixel values. The implementation in code follows this logic by marking the input batch as requiring gradients, running a forward and backward pass to obtain `images.grad`, taking the sign, and adding ϵ times this sign to the original images before clipping.

2.3 Projected Gradient Descent (PGD)

PGD extends FGSM to an iterative attack. Instead of taking a single step, the method takes several smaller steps of size α in the direction of the sign of the gradient, and projects the result back into the allowed L_∞ ball of radius ϵ around the original image after each step. Let $\tilde{x}^0 = \tilde{x}$ denote the original normalized input. For iteration $t = 0, 1, \ldots, T-1$, PGD performs

$$g^t = \nabla_{\tilde{x}} \mathcal{L}(\tilde{x}^t, y; \theta),$$

$$\tilde{x}^{t+1/2} = \tilde{x}^t + \alpha \cdot \operatorname{sign}(g^t),$$

$$\tilde{x}^{t+1} = \Pi_{B_\epsilon(\tilde{x}^0)}\big(\tilde{x}^{t+1/2}\big),$$

where $\Pi_{B_\epsilon(\tilde{x}^0)}$ denotes the projection onto the L_∞ ball around $\tilde{x}^0$ with radius ϵ. In coordinates, this projection is implemented as

$$\eta^{t+1} = \operatorname{clip}\big(\tilde{x}^{t+1/2} - \tilde{x}^0, -\epsilon, \epsilon\big), \quad \tilde{x}^{t+1} = \tilde{x}^0 + \eta^{t+1},$$

followed by clipping $\tilde{x}^{t+1}$ to the valid normalized data range. After T iterations, the final adversarial example is $\tilde{x}_{\text{adv}} = \tilde{x}^T$. The attack remains untargeted and uses the same loss $\mathcal{L}$ as in training.

Pseudocode: PGD attack

```
Input: model f, loss L, batch of images X, labels Y,
       epsilon, step size alpha, number of steps T
Output: batch of adversarial images X_adv

1. X_orig ← detach(X)              # save original images
2. X_cur  ← X_orig

3. For t = 1 to T do
4.     X_cur ← detach(X_cur); enable gradient for X_cur
5.     Z ← f(X_cur)               # forward pass
6.      ← L(Z, Y)
7.     Compute gradient G ←  / X_cur
8.     X_temp ← X_cur + alpha · sign(G)
9.      ← X_temp - X_orig
10.     ← clip(, -epsilon, epsilon)
11.    X_cur ← X_orig +
12.    X_cur ← clip(X_cur, X_min, X_max)
13. End for
```

```
14. X_adv ← detach(X_cur)
15. Return X_adv
```

In the implementation, the inner loop recreates the gradient graph at each step by detaching the current adversarial batch, marking it as requiring gradients, computing the loss, and backpropagating to obtain the gradient with respect to the current adversarial input. The step size α is chosen as a fraction of ϵ, so that several steps are needed to traverse the full ϵ ball, which makes the attack stronger than a single FGSM step.

2.4 Implementation setting

Both FGSM and PGD are applied to mini batches drawn from the CIFAR-10 test loader, after the classifier has been fully trained on the training split. All computations are done on normalized images, and gradients are taken with respect to these normalized inputs. For each attack, the adversarial batches $\tilde{x}_{\text{adv}}$ are passed through the fixed CNN to obtain predictions, which are later used to compute accuracy on adversarial data and the attack success rate as part of the experimental analysis.

3 Experiment

3.1 Setup

CNN Architecture

For this project, we trained a custom Convolutional Neural Network (SimpleCNN) designed for the CIFAR-10 dataset.

The architecture consists of:

Feature Extractor

- Conv2d($3 \to 32$), kernel size 3, padding 1 $\to$ ReLU $\to$ MaxPool(2×2)
- Conv2d($32 \to 64$), kernel size 3, padding 1 $\to$ ReLU $\to$ MaxPool(2×2)
- Conv2d($64 \to 128$), kernel size 3, padding 1 $\to$ ReLU $\to$ MaxPool(2×2)

Output feature map size: $128 \times 4 \times 4$

Classifier

- Flatten
- Linear($128 \times 4 \times 4 \to 256$) $\to$ ReLU
- Linear($256 \to 10$)

This SimpleCNN is relatively lightweight but expressive enough to handle the CIFAR-10 classification task.

Dataset Used

We use the CIFAR-10 dataset, which contains:

- 50,000 training images
- 10,000 test images
- Image size: 32×32 RGB
- 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck)

Data augmentation and normalization are applied:

Training Transformations

- RandomHorizontalFlip
- RandomCrop(32, padding=4)
- Normalization (mean = (0.4914, 0.4822, 0.4465), std = (0.2023, 0.1994, 0.2010))

Test Transformations

- Normalization only

Training Settings

- Optimizer: SGD
- Learning Rate: 0.01
- Momentum: 0.9
- Weight Decay: 5×10^{-4}
- Loss Function: CrossEntropyLoss
- Batch Size: 128
- Epochs: 40

The model was trained using PyTorch.

Model Performance on Clean Test Set

After completing training, the model was evaluated on the clean CIFAR-10 test set.

$$\text{Clean Test Accuracy} \approx 83\%$$

This accuracy serves as the baseline against which adversarial degradation is measured.

3.2 Evaluation

Both Fast Gradient Sign Method (FGSM) and Projected Gradient Descent (PGD) attacks are applied to evaluate model robustness under adversarial conditions.

3.2.1 Basic Evaluation

Clean Model Accuracy (ACC) The clean test accuracy reflects performance on unperturbed CIFAR-10 images. The model achieved a clean accuracy of 83.92%.

Attack Success Rate (ASR) ASR measures how effectively an attack forces incorrect predictions.

- **FGSM ($\epsilon = 0.05$):**
  - Adversarial accuracy drops sharply
  - ASR increases substantially
- **PGD ($\epsilon = 0.05$, $\alpha = \epsilon/4$, 10 steps):**
  - Even lower adversarial accuracy
  - Higher ASR than FGSM

Interpretation

- Even small perturbations ($0.03 \leq \epsilon \leq 0.05$) cause significant accuracy drops.
- FGSM already causes strong degradation.
- PGD, being iterative, is consistently more effective.
- Neural networks that perform well on clean data can be misled by imperceptible perturbations.

3.2.2 Advanced Evaluation

Varying FGSM Strength (ϵ) Evaluated at $\epsilon = 0.0, 0.01, 0.03, 0.05, 0.10$.
(refer figure 1)

Findings:

- Increasing ϵ leads to lower adversarial accuracy
- ASR increases almost monotonically
- At $\epsilon = 0.1$, accuracy collapses sharply

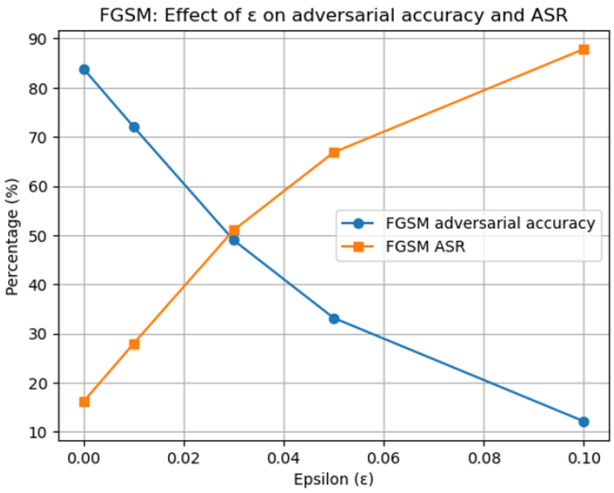

Figure 1: Varying FGSM strength

Varying PGD Steps Step counts tested: 5, 10, 20, 40 ($\epsilon = 0.05$, $\alpha = \epsilon/4$) (refer figure 2)

- More steps → better optimization
- Adversarial accuracy decreases with steps
- ASR rises accordingly

Performance Trends

- **Weaker attacks:** slight accuracy drop, barely visible perturbations
- **Stronger attacks:** near-complete accuracy collapse, still visually imperceptible

3.3 Visualization and Analysis

Clean vs FGSM vs PGD (refer figure 3)

- Clean images appear unchanged
- FGSM/PGD images look almost identical to clean
- Predictions differ drastically

Perturbation Heatmaps (refer figure 4, 5)

- FGSM: sharp, high-contrast perturbations
- PGD: smoother but more optimal perturbations

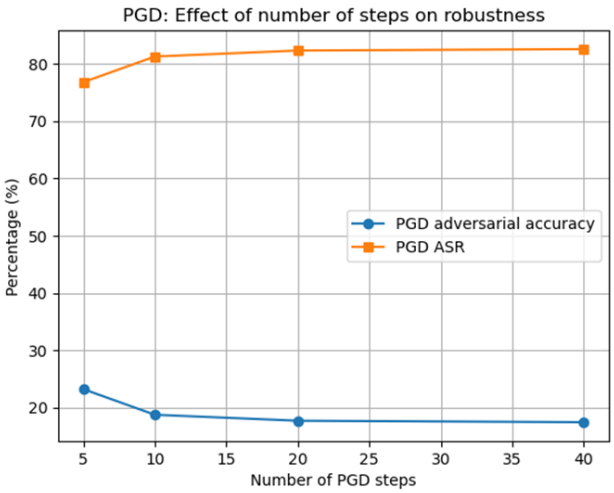

Figure 2: Varying PGD steps

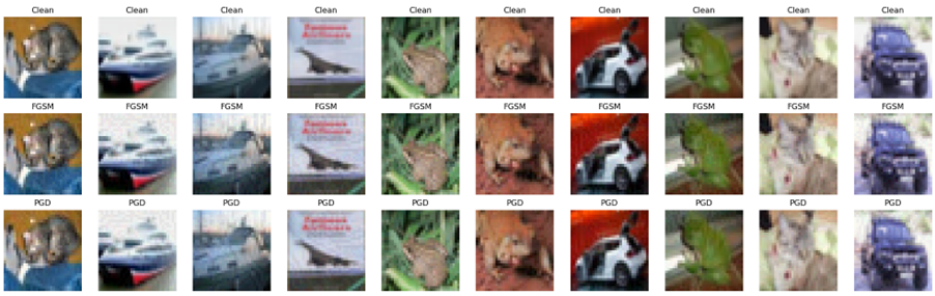

Figure 3: Clean vs FGSM vs PGD

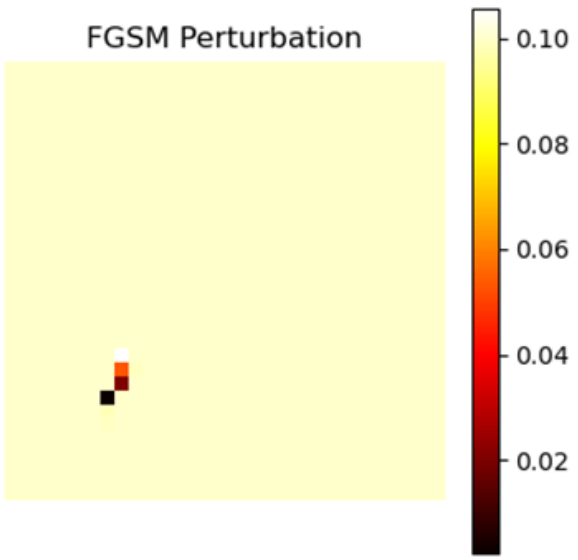

Figure 4: FGSM Perturbation

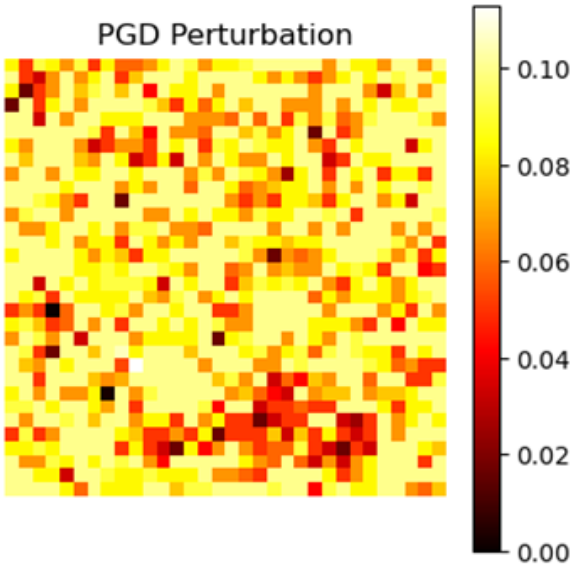

Figure 5: PGD Perturbation

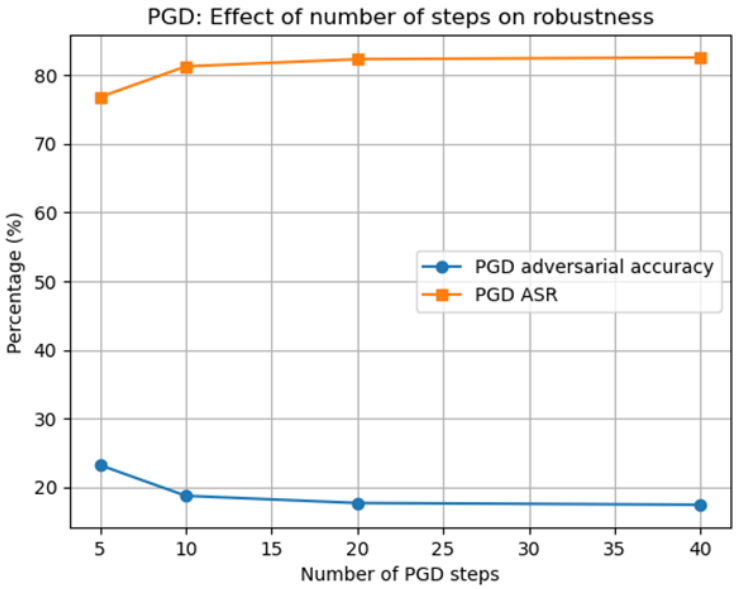

Figure 6: Line Plots of ACC and ASR

Line Plots of ACC and ASR (refer figure 6)

- Adversarial accuracy decreases with attack strength
- ASR increases correspondingly

3.4 Summary

- The CNN performs well on clean CIFAR-10 data.
- FGSM significantly reduces accuracy and increases ASR.
- PGD is a stronger attack and consistently outperforms FGSM.
- Increasing ϵ or PGD steps results in stronger attack performance.
- Visualizations confirm perturbations are nearly invisible yet highly destructive.
- Without adversarial training, robustness is very low.

4 Contribution

- Aayush Kushwaha - Refactoring the code, carrying out the final implementation and integration of the system, producing the visualizations, and performing the advance evaluation.
- Balaka Biswas - Research on CNN and white-box attacks, literature review, and undertaking the 'Objective', 'Methodology', and 'Challenges and Solutions' sections of the project report.
- Harsh Deshmukh - Developed the initial code draft, implemented core modules, performed the evaluation and carried out initial testing.
- Sahil Khandu Thorat - Undertaking the Experiments section of the project report, designing and executing the experimental setup, performing dataset preprocessing and parameter tuning, and analyzing the experimental results for the final evaluation.

5 Challenges and Solutions

5.1 Computational resources and runtime

A first challenge in the project was the computational cost of training the CNN and running multiple adversarial evaluations. Training the model for several epochs on CIFAR-10, with data augmentation and three convolutional blocks, is already time-consuming on a CPU. The cost increased further when FGSM and especially PGD were added, since both attacks require extra forward and backward passes through the network. For PGD, each step in the attack loop performs a full gradient computation with respect to the input, and this is repeated for several iterations and for many mini batches. Running all of this on a CPU made it difficult to complete training and the full set of attack experiments in a reasonable amount of time.

To address this, the group moved the training and evaluation code to a machine with a GPU. The PyTorch code was configured to use the `device` abstraction so that the model and all input batches were transferred to `cuda` when a GPU was available. In practice, this meant running the project on Aayush's Macbook, which had GPU support available for PyTorch. Once the CNN and the adversarial attack loops were executed on the GPU, the time per epoch decreased noticeably. This made it feasible to train the network with data augmentation, to evaluate FGSM for multiple values of ϵ, and to run PGD with different numbers of steps, without having to reduce the size of the test subset or the number of attack configurations.

5.2 Image generation and visualization

A second challenge was how to generate and visualize images that made the effect of the attacks clear. During training and evaluation, all CIFAR-10 images are normalized channel-wise using the dataset mean and standard deviation. If these normalized tensors are plotted directly, they appear washed out or clipped and do not resemble the original dataset. It was also important to show clean, FGSM, and PGD images side by side in a consistent layout and to highlight the perturbations themselves through some kind of heatmap, without introducing extra distortions.

The solution was to add a small set of utility functions that handle image reconstruction and plotting. After the model finishes training, a fixed inverse normalization transform is defined for the three channels, using the same μ_c and σ_c as in preprocessing. When an image tensor $\tilde{x} \in \mathbb{R}^{3\times32\times32}$ is ready to be displayed, the code applies the inverse transform channelwise,

$$x_c = \sigma_c \tilde{x}_c + \mu_c, \quad c \in \{1, 2, 3\},$$

converts the result from $[C, H, W]$ to $[H, W, C]$, and clips values to the range $[0, 1]$. This reconstructed array is then passed to `matplotlib` for plotting. A helper function wraps these steps so that clean images, FGSM images, and PGD images can be displayed in aligned grids, either as three rows for a fixed set of examples or as three columns per row.

For perturbation visualization, the code computes the difference between an adversarial image and its corresponding clean image. Given a clean tensor $\tilde{x}$ and an adversarial tensor $\tilde{x}_{\text{adv}}$, the perturbation is

$$\Delta = \tilde{x}_{\text{adv}} - \tilde{x},$$

and a single channel heatmap is formed by taking the mean absolute value across the three channels,

$$H(i, j) = \frac{1}{3} \sum_{c=1}^{3} \big|\Delta_c(i, j)\big|.$$

This two-dimensional array H is then shown using a `hot` colormap, with a color bar to indicate relative magnitude. In addition, the code stores a small collection of adversarial images for each value of ϵ in dictionaries, so that the same examples can be reused for different plots without regenerating them. Together, these choices made it possible to present clean, FGSM, and PGD images that remain visually close to the original CIFAR-10 samples, while also providing clear visual evidence of where the attacks change the pixel values.