Hands-On Labs in Machine Learning and Cybersecurity: Practical Experience in AI Threat Mitigation

Vinay Kumar Posina, Ajay Kumar Reddy Poreddy, Monish Kamal Bathini, Deekshith Machidi

TABLE I: List of Key Notations

Symbol	Description
X	Original input image
X_0	Initial adversarial example, set to X
X_1'	Perturbed image after one iteration
α	Step size for perturbation
$\text{sign}(\cdot)$	Element-wise sign function
$\nabla_x J(X_0, Y_{\text{true}})$	Gradient of loss J for X_0 and label Y_{true}
ϵ	Maximum allowed perturbation per pixel
$\min(255, \max(0, X - \epsilon, X_1'))$	Clamps pixels within bounds and ϵ range of X
x	Input image
x'	Adversarial example generated from x
L_0 distance	Count of modified pixels, $x_i \neq x_i'$
L_2 distance	Euclidean distance between x and x'
L_∞ distance	Maximum per-pixel perturbation
λ	Trade-off between misclassification and perturbation size
η	Perturbation added to x
$\nabla_x J(\theta, x, y)$	Gradient of loss J with respect to x and label y
$x = x + \eta$	Adversarial example after adding perturbation η
g_t	Gradient at iteration t
μ	Decay factor for previous gradient
g_{t-1}	Previous gradient
$\nabla_x J(x_t, y)$	Gradient of J for x_t and label y
x_t	Adversarial example at iteration t
$x_{t+1} = x_t + \alpha \cdot \text{sign}(g_t)$	Update rule for next iteration
δ	Perturbation applied to x
$\|\delta\|_p \leq \epsilon$	Limits L_p-norm of δ to ϵ
$x'_{t+1} = \Pi_{B(\epsilon)}(x)(x'_t + \alpha \cdot \text{sign}(\nabla_{x'} L(\theta, x'_t, y)))$	Iterative update with projection
$\text{clip}(x', a, b)$	Clips x' within $[a, b]$
x_0'	Initial adversarial example with random δ
δ	Random perturbation in $[-\epsilon, \epsilon]$
$U(-\epsilon, \epsilon)$	Uniform distribution for δ
input $- \epsilon$	Lower bound for pixel values
input $+ \epsilon$	Upper bound for pixel values

I. Introduction

The rapid advancements in machine learning (ML) and artificial intelligence (AI) have transformed numerous industries, offering unprecedented opportunities for innovation and efficiency. However, these technologies have also introduced new challenges, particularly in the realm of cybersecurity. As the capabilities of AI systems [1] grow, so do the threats they face, including adversarial attacks, data breaches, and model exploitation. Addressing these challenges requires practical experience and hands-on experimentation to understand, detect, and mitigate AI-driven threats.

This paper presents *Hands-On Labs in Machine Learning and Cybersecurity*, an initiative aimed at bridging the gap between theoretical knowledge and practical application in the fields of AI and cybersecurity. By integrating real-world scenarios and hands-on experiments, this project empowers learners and professionals to tackle AI-related security challenges effectively.

The project focuses on three key objectives:

1) **Developing Expertise in Machine Learning Security**: Understanding how machine learning models can be attacked, manipulated, or exploited, and devising strategies to safeguard them.
2) **Building Robust Models Through Adversarial Training**: Equipping participants with practical tools and techniques, such as adversarial training, to enhance model robustness against attacks and ensure secure AI deployments.
3) **Exploring Federated Learning for Decentralized Security**: Investigating the vulnerabilities and resilience of federated learning systems, where decentralized data processing introduces unique challenges, such as secure aggregation, communication efficiency, and privacy preservation.

By addressing these objectives, the initiative not only enhances understanding of ML security but also explores the potential of federated learning as a paradigm to promote privacy-aware and robust AI systems. Federated learning, with its decentralized approach to model training across distributed data sources, presents both opportunities for enhancing privacy and new attack surfaces that require specialized mitigation strategies.

II. Datasets

In this study, we utilized two widely recognized datasets: FashionMNIST and CIFAR-10. These datasets were chosen due to their diversity, relevance, and widespread use in benchmarking machine learning and deep learning models.

A. *CIFAR-10 (Canadian Institute For Advanced Research)*

The CIFAR-10 dataset is a widely used benchmark in machine learning and computer vision research. It comprises 60,000 color images, each sized at 32×32 pixels, categorized into 10 distinct classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Each class contains 6,000 images, with 50,000 allocated for training and 10,000 for testing. Developed by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton, CIFAR-10 serves as a standard dataset for evaluating

image recognition algorithms. Its manageable size and diversity make it ideal for developing and testing machine learning models, particularly in the realm of image classification. [2]

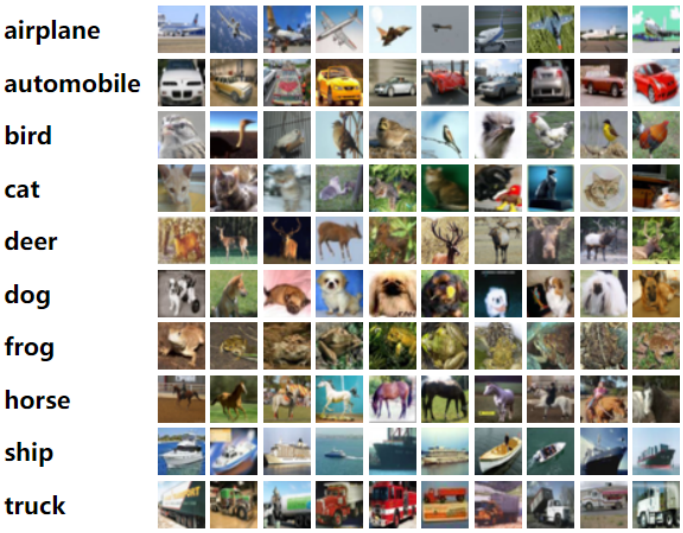

Fig. 1: Sample images from the CIFAR-10 dataset, illustrating the 10 distinct classes. [3]

B. Fashion-MNIST Dataset

The Fashion-MNIST(Modified National Institute of Standards and Technology database) dataset is a popular benchmark in machine learning and computer vision, designed as an upgraded alternative to the well-known MNIST dataset of handwritten digits. Introduced by Xiao et al. [4], this dataset contains grayscale images of fashion-related items such as clothing and accessories. Fashion-MNIST is often used to test image classification models, offering a more diverse and challenging task compared to MNIST while maintaining a similar structure.

Fashion-MNIST includes a total of 70,000 grayscale images, each sized 28×28 pixels, categorized into 10 unique classes. The dataset is split into: Training set: 60,000 images for model training Test set: 10,000 images for evaluating model performance.

Each image contains one fashion item, properly centered and scaled to fit the 28×28 grid. The pixel values range from 0 to 255, which can be normalized for better model performance.

Fashion-MNIST consists of 10 distinct categories, each labeled with an integer from 0 to 9:

- T-shirt/Top
- Trouser
- Pullover
- Dress
- Coat
- Sandal
- Shirt
- Sneaker
- Bag
- Ankle Boot

Each image belongs to one of these categories, making the dataset well-suited for single-label classification tasks.

Both datasets provided a strong foundation for testing and validating the proposed methodologies. They allowed us to examine model generalization across different types of visual data, from grayscale fashion images to colorful and complex objects in CIFAR-10. [4]

Fig. 2: Sample images from the Fashion-MNIST dataset, illustrating the 10 distinct classes. [5]

III. Adversarial attacks.

Adversarial attacks are a class of techniques aimed at deceiving machine learning models by introducing subtle, often imperceptible perturbations to the input data. These attacks exploit vulnerabilities in the model's decision boundaries, resulting in misclassifications or incorrect predictions. [6] [7]

Adversarial attacks can be broadly categorized into:

- **White-box attacks:** The attacker has full knowledge of the model, including its architecture, parameters, and training data. [8]
- **Black-box attacks:** The attacker has no prior knowledge of the model but can query it to generate adversarial examples. [8]

Common methods for generating adversarial examples include:

- **Fast Gradient Sign Method (FGSM):** Introduced by Goodfellow et al. [7], this method computes adversarial perturbations by maximizing the model's loss function with respect to the input data.
- **Projected Gradient Descent (PGD):** An iterative extension of FGSM, PGD applies repeated small perturbations within a bounded range to generate stronger adversarial examples [9].

- **Carlini and Wagner (C&W) Attacks:** A more targeted approach that minimizes the distortion while ensuring misclassification [10].
- **Basic Iterative Method (BIM):** A method that iteratively applies FGSM with smaller step sizes, refining perturbations for higher attack success while staying within the allowed perturbation limit [11].

Adversarial attacks are particularly concerning in critical applications such as healthcare, autonomous driving, and cybersecurity, where misclassifications can have severe consequences. Understanding and mitigating these vulnerabilities is a critical area of research in AI security.

A. *Fast Gradient Signed Method (FGSM)*

There are two types of adversarial attacks: White-box attacks and Black-box attacks. White-box attacks have the complete knowledge of the targeted model, including its parameter values, architecture, training method, and in some cases its training data as well. Black-box attacks feed a targeted model with the adversarial examples (during testing) that are generated without the knowledge of that model. By observing the output of the targeted model and adjusting the parameters, black-box attacks could also attack many neural network based model. [12]

As FGSM is a type of white box attack, the attacker needs to have knowledges of the pretrained model, different model will have different parameters and loss function.

$$\tilde{x} = x + \eta \tag{1}$$

In Equation 1, $\tilde{x}$ is the adversarial sample, x is the original input, the η is the adversarial perturbation added to the original input.

$$\eta = \epsilon \times \text{sign}(\nabla_x J(\theta, x, y)) \tag{2}$$

In Equation 2, y is the targets associated with x. In face recognition scenario, x, y are face images, which x is our input images and y is the victim image(the targeting image). J is the loss function of the current model, θ is the set of parameters in this model. In this case the cofficient ϵ indicates the size of the step that perturbation moving backwards at the direction of the gradients. By changing the ϵ we can control how much perturbation adding to the input images. And this "step" can also divided into several smaller steps, which is referred as granularity of the perturbation.

B. *Carlini-Wagner Attack (CW-l2)*

The Carlini-Wagner attack is a significant method in adversarial machine learning aimed at generating adversarial examples that can mislead neural networks. The attack is based on finding small perturbations to input data that cause misclassification while keeping the changes imperceptible. Carlini and Wagner's method is highly effective against networks, even those with some defenses like defensive distillation. [10]

The research focuses on optimizing the adversarial examples with respect to different distance metrics, such as the L_2 norm, L_0 norm, and L_∞ norm, which measure the level of perturbation. These metrics help generate adversarial examples that evade detection while remaining close to the original input data. This method has been used extensively to test the robustness of models used in image classification tasks.

In more detail:

- L_0 distance measures the number of coordinates i such that $x_i \neq x'_i$. Thus, the L_0 distance corresponds to the number of pixels that have been altered in an image.
- L_2 distance measures the standard Euclidean (root-mean-square) distance between x and x'. The L_2 distance can remain small when there are many small changes to many pixels.
- L_∞ distance measures the maximum change to any of the coordinates:

$$\|x - x'\|_\infty = \max\left(|x_1 - x'_1|, \ldots, |x_n - x'_n|\right)$$

The Carlini-Wagner attack optimizes a specific loss function that balances the trade-off between the perturbation size and the model's misclassification. The loss function generally has two components:

- **Misclassification Loss:** This term ensures that the perturbed image is classified incorrectly by the target model. It typically involves adding a term that penalizes correct classifications and encourages misclassifications.
- **Perturbation Loss:** This term controls the magnitude of the perturbation, and it is based on the chosen norm (e.g., L_2, L_∞, or L_0). It ensures that the perturbation is minimized according to the specific norm being used.

The general form of the loss function for the Carlini-Wagner attack is:

$$\text{Loss} = \text{Misclassification Loss} + \lambda \cdot \text{Perturbation Loss}$$

where λ is a trade-off parameter that balances the importance of the perturbation size versus the misclassification.

C. *Projected Gradient Descent (PGD)*

The Projected Gradient Descent (PGD) attack operates as a white-box attack, granting the attacker access to both the model's gradients and its weights. This privileged access significantly boosts the attacker's ability, compared to black-box attacks, as it allows them to fine-tune their approach to mislead the model without relying on transfer attacks, which often result in perceptible distortions. PGD is widely recognized as the most robust white-box adversarial method, as it imposes no restrictions on the time or effort the attacker can allocate to crafting the most successful attack. PGD iteratively updates the adversarial example x' by computing gradients and projecting the updates onto the allowed perturbation space.

The goal of adversarial attacks is to find a small perturbation δ that, when added to the original input x, causes the model to misclassify $x + \delta$:

$$\begin{aligned} &\text{Find } \delta \text{ such that} \\ l \quad & x' = x + \delta \\ & \text{subject to } \|\delta\|_p \leq \epsilon, \end{aligned}$$

The iterative update rule for PGD is as follows:

$$x'_{t+1} = \Pi_{\mathcal{B}_\epsilon(x)} \left(x'_t + \alpha \cdot \text{sign} \left(\nabla_{x'} L(\theta, x'_t, y) \right) \right),$$

- $\Pi_{\mathcal{B}_\epsilon(x)}(x')$ denotes the projection of x' onto the L_∞ norm ball centered at x with radius ϵ,
- $\text{clip}(x', a, b)$ clips each element of x' to be within the interval $[a, b]$,
- This operation maintains the perturbation within the allowed norm constraint.

The attack is initialized by adding a random perturbation within the allowed norm ball:

$$x'_0 = x + \delta, \quad \delta \sim \mathcal{U}(-\epsilon, \epsilon),$$

where:

- x'_0 is the initial adversarial example,
- δ is a random perturbation sampled uniformly,
- $\mathcal{U}(-\epsilon, \epsilon)$ denotes a uniform distribution in the range $[-\epsilon, \epsilon]$.

This random start helps in escaping certain non-robust local minima and strengthens the attack.

D. *Basic Iterative Method (BIM)*

The Basic Iterative Method (BIM), also known as the Iterative FGSM, is an extension of the Fast Gradient Sign Method (FGSM) for generating adversarial examples. Proposed by Kurakin et al. [11], BIM applies FGSM iteratively with smaller step sizes to refine the adversarial perturbations. This iterative approach increases the effectiveness of the attack by allowing the perturbations to converge more precisely toward a successful adversarial example.

The update rule for BIM is given as:

$$x_{t+1} = \text{Clip}_{x,\epsilon} \left(x_t + \alpha \cdot \text{sign}(\nabla_x J(x_t, y)) \right), \tag{3}$$

where:

- x_t: Adversarial example at iteration t,
- x_{t+1}: Adversarial example at iteration $t + 1$,
- α: Step size for each iteration,
- ϵ: Maximum perturbation allowed (norm constraint),
- $J(x, y)$: Loss function of the model for input x and true label y,
- $\text{Clip}_{x,\epsilon}(\cdot)$: A clipping function to ensure the perturbation stays within the $[x - \epsilon, x + \epsilon]$ range.

The method begins with the original input $x_0 = x$ and iteratively applies FGSM updates. The clipping operation ensures the adversarial perturbation remains imperceptible and constrained within the allowed budget ϵ.

1) Advantages of BIM: Compared to a single-step FGSM, BIM generates stronger adversarial examples that are more likely to fool machine learning models. The iterative nature of BIM allows for:

- Higher success rates in fooling models.
- Improved adversarial example quality with smaller perturbations.

However, BIM is computationally more expensive due to the multiple iterations required to generate an adversarial example.

IV. Overview of ResNet-18

ResNet-18 [13] is a convolutional neural network designed to address challenges in training deep architectures, such as vanishing gradients. It employs residual connections, enabling efficient learning even with increased depth. The architecture consists of 18 layers structured into four stages, each containing residual blocks. Its computational efficiency and high performance make it a popular choice for various image classification and feature extraction tasks.

A. *Architecture Details*

The structure of ResNet-18 is outlined in Table II, showing how the spatial dimensions decrease while feature richness increases. Skip connections in residual blocks facilitate the learning process by focusing on residual mappings. [13] [14]

Stage	Layer Type	Output Size	Details
Input	Conv2D + BN + ReLU	112×112	7×7 conv, 64 filters, stride 2
	MaxPooling	56×56	3×3 pooling, stride 2
1	Residual Block	56×56	2 blocks, 64 filters
2	Residual Block	28×28	2 blocks, 128 filters, stride 2 in 1st block
3	Residual Block	14×14	2 blocks, 256 filters, stride 2 in 1st block
4	Residual Block	7×7	2 blocks, 512 filters, stride 2 in 1st block
Output	Global Avg Pooling	1×1	Reduces to a single value per filter
	Fully Connected (FC)	N/A	Class predictions (e.g., 1000 classes)

TABLE II: ResNet-18 Architecture Overview

B. *Residual Learning and Skip Connections*

The innovation in ResNet-18 lies in its use of residual blocks. Each block comprises two 3×3 convolutional layers with batch normalization and ReLU activation. The skip connection bypasses these layers, adding the input of the block directly to its output, as shown in Fig. 3.

$$y = F(x) + x \tag{4}$$

where x is the input, $F(x)$ is the output of the convolutional layers, and y is the final output of the block.

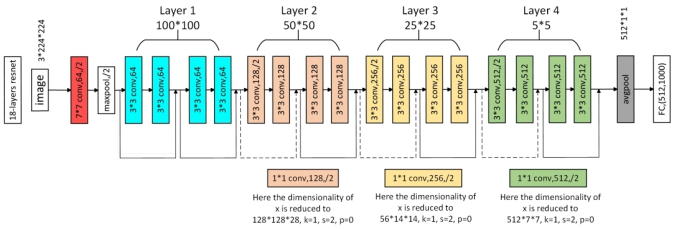

Fig. 3: ResNet-18 Architecture [15]

C. *Parameters and Computational Complexity*

ResNet-18 contains approximately 11.7 million parameters, distributed across its layers. The use of smaller filters (3×3) and a streamlined architecture minimizes computational overhead, making it suitable for both high-performance and resource-constrained environments.

D. *Applications*

ResNet-18 has found applications in:

- **Image Classification**: Achieving high accuracy on ImageNet and other datasets.
- **Transfer Learning**: Adapting pre-trained models for custom tasks, such as medical imaging.
- **Feature Extraction**: Using intermediate layers for tasks like object detection.

V. Federated Learning Architecture

Federated Learning (FL) is an innovative approach to decentralized machine learning that enables multiple devices to collaborate on training a shared model without exchanging raw data. By keeping the data on local devices, this technique addresses privacy concerns while utilizing distributed computational resources. Below, we outline the key components and workflow of the FL architecture. [16]

A. *Key Components*

The architecture of federated learning is built around the following main elements:

- **Client Devices (Participants):** FL involves multiple client devices—such as smartphones, IoT devices, or edge servers—that hold local datasets. These devices independently train a global model on their data, sending updates back to a central server without transferring the raw data. [18]
- **Central Server (Aggregator):** A central server orchestrates the entire learning process. It collects updates from the clients, aggregates them using algorithms like Federated Averaging (FedAvg), and maintains the global model. [18]
- **Global Model:** This shared model starts with an initial configuration provided by the central server and evolves as clients contribute their updates. The central server periodically redistributes the improved model back to the clients. [19]
- **Communication Protocol:** Secure communication channels facilitate the exchange of model updates between clients and the server. Techniques like model compression and sparsity are used to reduce communication costs. [20]
- **Local Training:** Each client trains the global model using its local dataset. The process is powered by optimization algorithms, such as stochastic gradient descent (SGD), for a set number of epochs. The trained parameters are then sent to the server. [17]
- **Aggregation and Synchronization:** The server aggregates updates from all the clients to update the global model. This ensures that all clients start the next round of training with the same version of the model. [18]

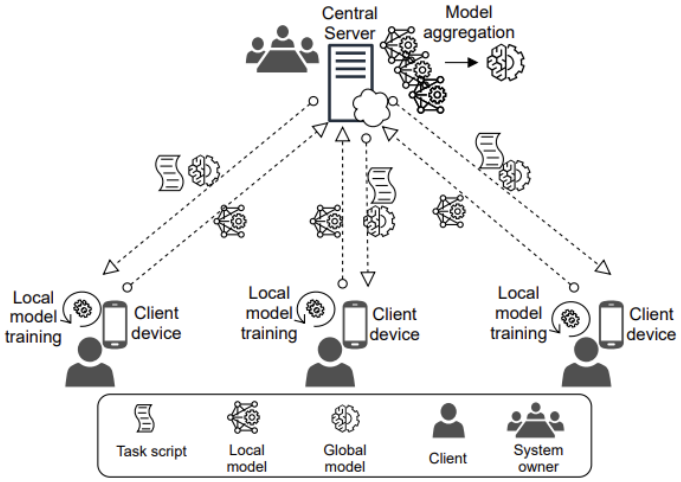

Fig. 4: Federated Learning Architecture [17]

B. *Workflow of Federated Learning*

The federated learning process operates iteratively in several steps:

1) **Model Initialization:** The central server initializes the global model and sends it to participating clients. [16]
2) **Local Training:** Clients train the model on their local datasets and optimize the model's parameters.
3) **Model Updates Transmission:** Clients send the updated model parameters (such as gradients or weights) to the central server while keeping their data private.
4) **Global Aggregation:** The server aggregates the updates received from clients to create a new global model.
5) **Model Distribution:** The server redistributes the refined global model back to the clients for the next round of training.
6) **Convergence:** This iterative process continues until the global model achieves satisfactory performance or convergence.

C. *Benefits of Federated Learning*

Federated learning offers several advantages over traditional centralized approaches:

- **Data Privacy:** Local data remains on devices, ensuring user privacy.
- **Scalability:** FL supports large-scale, distributed networks with multiple clients.
- **Personalization:** Models can be customized for individual users or devices.
- **Efficiency:** It minimizes the need for centralized data transfer and storage, reducing bandwidth usage.

‘

D. *Applications and Impact*

This architecture has been applied in domains such as healthcare, IoT, and mobile device personalization, showcasing its potential to address modern data privacy challenges while enabling distributed machine learning. Federated learning has opened new avenues for innovation in data-sensitive fields, providing a scalable and secure solution for collaborative intelligence. [21]

VI. Adversarial Training

Adversarial training improves deep learning model robustness against adversarial attacks by augmenting the training dataset with adversarial examples. These modified inputs are designed to mislead the model. By incorporating these examples into training, the model learns to recognize and correctly classify adversarial inputs, boosting resilience to attacks. [22]

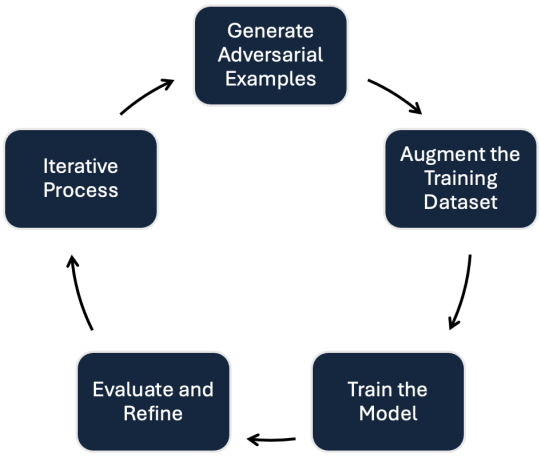

Fig. 5: Adversarial Training Workflow

A. *The key steps in adversarial training typically involve:*

1) **Generate Adversarial Examples:** For each training sample, generate an adversarial example by adding a small perturbation that maximizes the model's loss for that sample. This perturbation is crafted using methods like FGSM (Fast Gradient Sign Method) or PGD (Projected Gradient Descent).
2) **Inner Maximization (Attack Step):** Solve the inner maximization problem to create adversarial examples that maximize the loss function for the current model parameters. This step finds the worst-case perturbations within a predefined limit, typically bounded by an ℓ_p norm.
3) **Outer Minimization (Training Step):** Adjust the model parameters to minimize the loss on both clean and adversarial examples. This step updates the model to be more resilient to adversarial attacks.
4) **Evaluate Robustness:** Finally, evaluate the adversarial robustness by testing the model on various adversarial attacks and assessing its accuracy on perturbed samples.
5) **Repeat:** Repeat steps 1-3 for each batch and epoch during training until the model achieves desired robustness or the training converges.

The general adversarial training objective is to minimize the loss over both clean and adversarial examples:

$$L_{\text{train}} = \mathbb{E}_{(x,y)\sim D}\left[\mathcal{L}(f(x), y)\right] + \mathbb{E}_{(x',y)\sim D_{\text{adv}}}\left[\mathcal{L}(f(x'), y)\right],$$

where:

- x is the clean input,
- x' is the adversarial example generated from x,
- $f(x)$ is the model's prediction for input x,
- $\mathcal{L}(f(x), y)$ is the loss function used for training,
- D is the distribution of clean data, and
- D_{adv} is the distribution of adversarially perturbed data.

Adversarial training aims to minimize the total loss across both clean and adversarial examples, allowing the model to develop robustness to adversarial perturbations.

B. *Challenges and Limitations*

While adversarial training improves robustness, it comes with several challenges:

- **Computational Cost:** Generating adversarial examples and training the model on these examples is computationally expensive. It requires additional time and resources, especially for large datasets and deep models.
- **Overfitting to Attacks:** If the adversarial examples are not carefully selected or if the training is not diverse enough, the model might overfit to specific types of attacks. This limits the model's generalization ability to other unseen attacks.
- **Trade-Off with Clean Accuracy:** Adversarial training typically leads to a decrease in the model's accuracy on clean (non-adversarial) examples. Thus, there is a trade-off between improving robustness and maintaining performance on standard test data.
- **Robustness to Diverse Attacks:** The model might become robust only to the specific attack method used during adversarial training. Thus, it might still be vulnerable to other types of attacks or more sophisticated variants of adversarial examples.

Adversarial training is one of the most effective techniques to enhance neural network robustness against adversarial

attacks. Researchers are exploring ways to improve its efficiency and generalization, such as through advanced adversarial generation techniques, better regularization strategies, and multi-step adversarial training. [23]

VII. Experimental Setup

In this section, we outline the experimental setup employed to evaluate the robustness and performance of our models under both standard and adversarial conditions. Our experiments are conducted on two widely recognized datasets—FashionMNIST and CIFAR-10—using Convolutional Neural Networks (CNN) and ResNet-18 architectures. We also simulate a federated learning environment with a mix of normal and adversarial clients to assess the impact of adversarial attacks in a distributed setting. [24]

A. Datasets

- **FashionMNIST**: This dataset comprises 70,000 grayscale images of size 28×28 pixels, categorized into 10 classes representing different clothing items. We use 60,000 images for training and 10,000 for testing. [25]
- **CIFAR-10**: Consisting of 60,000 color images of size 32×32 pixels across 10 classes, CIFAR-10 is split into 50,000 training images and 10,000 test images. [26]

B. Models

- **Convolutional Neural Network (CNN)**: We design a standard CNN architecture tailored for each dataset, featuring multiple convolutional layers followed by pooling and fully connected layers. [27]
- **ResNet-18**: A residual network with 18 layers, ResNet-18 incorporates skip connections to mitigate the vanishing gradient problem, facilitating the training of deeper networks. [28]

C. Adversarial Attacks

To evaluate model robustness, we implement the following adversarial attack methods:

- **Fast Gradient Sign Method (FGSM)**: Generates adversarial examples by adding perturbations in the direction of the gradient of the loss with respect to the input.
- **Projected Gradient Descent (PGD)**: An iterative version of FGSM that applies multiple small perturbations while projecting the adversarial examples back onto the permissible data space.
- **Carlini & Wagner Attack**: An optimization-based attack that minimizes the perturbation required to misclassify inputs, often bypassing defensive distillation.
- **Basic Iterative Method (BIM)**: An extension of FGSM that applies the attack iteratively with smaller step sizes to refine adversarial examples.

For the federated learning simulations, we focus on the FGSM attack to assess the impact of adversarial clients.

D. Federated Learning Configuration

We simulate a federated learning environment with 10 clients:

- **Normal Clients**: 6 clients participate honestly, training on their local data without manipulation.
- **Adversarial Clients**: 4 clients perform label flipping attacks, where 50% of the labels in their local datasets are intentionally misassigned to incorrect classes.

Data distribution among clients is uniform, ensuring each client has an equal share of the dataset. The federated learning process involves multiple communication rounds, where each client trains the model locally and sends updates to a central server for aggregation using federated averaging.

E. Hardware and Software Environment

- **Hardware**: Experiments are conducted on a workstation equipped with an NVIDIA Quadro RTX 5000 GPU, providing the necessary computational power for training deep learning models.
- **Software**: We utilize PyTorch as the primary deep learning framework, leveraging its capabilities for model building and training in both centralized and federated settings.

F. Training Parameters

- **Learning Rate**: Set to an initial value of 0.01, with a decay schedule applied every 10 epochs.
- **Batch Size**: A batch size of 64 is used for both local and federated training.
- **Optimization Algorithm**: Stochastic Gradient Descent (SGD) with momentum is employed to optimize the model parameters.
- **Number of Epochs**: Models are trained for 50 epochs in centralized settings and 100 communication rounds in federated learning.

G. Evaluation Metrics

Model performance is assessed using the following metrics:

- **Accuracy**: The proportion of correctly classified instances over the total instances. [29]
- **Robustness Against Adversarial Attacks**: Measured by the model's accuracy on adversarially perturbed test sets generated using the aforementioned attack methods. [30]

H. Adversarial Attack Simulation

To simulate adversarial conditions, we generate adversarial examples using FGSM with varying perturbation magnitudes (ϵ values). These adversarial examples are used to test the robustness of the trained models in both centralized and federated learning setups.

By meticulously configuring the experimental parameters and incorporating adversarial scenarios, we aim to comprehensively evaluate the resilience and reliability of our models in realistic environments. [31]

VIII. Results

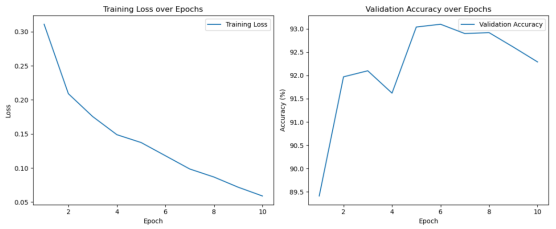

Fig. 6: Analysis of without attack Model Performance

Training Loss over Epochs: The training loss shows a steady decline as the model trains, starting around 0.30 at the first epoch and gradually dropping to about 0.05 by epoch 10. This suggests that the model is successfully learning and improving its ability to make predictions over time.

Validation Accuracy over Epochs: The validation accuracy generally increases from around 89.5% to approximately 92% by epoch 10. However, there are some fluctuations along the way, with a noticeable peak at epoch 6, which indicates a moment where the model performed particularly well. These fluctuations suggest that the model's ability to generalize to unseen data may not be entirely stable.

Analysis: The consistent drop in training loss and the increase in validation accuracy indicate that the model is learning effectively and improving in its ability to generalize to new data. While the overall trend is positive, the fluctuations in validation accuracy hint that there might be room for further refinement, such as fine-tuning the model or addressing overfitting.

A. Fast Gradient Sign Method (FGSM) Results

The results of the FGSM attack are shown in Figure 7. The original image is correctly classified as "cock" (label 7). However, as the perturbation magnitude (ϵ) increases, the classifier becomes progressively less accurate, demonstrating its vulnerability to adversarial noise.

- $\epsilon = 0.02$**:** The image is misclassified as "partridge" (label 86), despite minimal noise.
- $\epsilon = 0.05$**:** The model continues to misclassify the image as "partridge."
- $\epsilon = 0.1$ **and** $\epsilon = 0.15$**:** With moderate noise, the model predicts "scarf" (label 824).
- $\epsilon = 0.2$ **and** $\epsilon = 0.25$**:** Higher perturbation levels lead to misclassification as "bubble" (label 971).

These results highlight the susceptibility of deep learning models to adversarial attacks, even when perturbations are imperceptible to humans. This emphasizes the need for robust defenses like adversarial training and noise regularization. [9]

Fig. 7: Fast Gradient Signed Method

B. Carlini-Wagner (CW) Attack Results

The results of the Carlini-Wagner (CW) attack are presented in Figure 8. The CW attack generates adversarial perturbations that are visually imperceptible while effectively deceiving the classifier.

- **Original Image**: The image is correctly classified with a prediction of "339" (e.g., Horse).
- **Adversarial Image**: After applying the CW attack, the image remains visually indistinguishable to humans but is misclassified by the model with a prediction of "351" (a different class). [10]

Fig. 8: Carlini-Wagner Attack

The CW attack demonstrates the ability to craft adversarial examples that are not only effective in deceiving classifiers but also visually inconspicuous, making them challenging to detect with traditional defense mechanisms.

C. Projected Gradient Descent (PGD) Attack Results

The results of the Projected Gradient Descent (PGD) attack are shown in Figure 9. PGD generates adversarial perturbations iteratively, refining the attack to mislead the classifier while keeping the perturbation imperceptible to humans.

- **Original Image**: The image is correctly classified as "fox squirrel" with label 335.
- **Adversarial Image**: After applying the PGD attack, the image is visually similar to the original but is misclassified by the model as "baboon" with label 372.

These results highlight the effectiveness of PGD in generating adversarial examples that are difficult to detect but successfully deceive the classifier. This underscores the importance of adversarial defense mechanisms in deep learning models. [32]

Fig. 9: Projected Gradient Descent Attack

D. Basic Iterative Method (BIM) Attack Results

The results of the Basic Iterative Method (BIM) attack are presented in Figure **??**. BIM applies adversarial perturbations iteratively, refining the noise at each step to effectively deceive the classifier while maintaining visual similarity to the original image.

- **Original Image**: The image is correctly classified as "fox squirrel" with label 335.
- **Adversarial Image**: After applying the BIM attack, the image is nearly indistinguishable from the original to the human eye but is misclassified by the model as "baboon" with label 372.

These results demonstrate BIM's capability to generate effective adversarial examples while preserving visual similarity, posing a challenge for detecting and defending against such attacks. [33]

Fig. 10: Basic Iterative Method

Attack Type	Perturbation Visibility	Model Fooling Rate	Computational Complexity
C&W	Minimal	High	High
FGSM	Scalable (based on epsilon)	Moderate to High	Low
PGD	Moderate	High	High
BIM	Moderate	High	Moderate

TABLE III: Comparison of Adversarial Attack Types

References

[1] I. Rosenberg, A. Shabtai, Y. Elovici, and L. Rokach, "Adversarial machine learning attacks and defense methods in the cyber security domain," *ACM Computing Surveys (CSUR)*, vol. 54, no. 5, pp. 1–36, 2021.

[2] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 and cifar-100 datasets," *URl: https://www. cs. toronto. edu/kriz/cifar. html*, vol. 6, no. 1, p. 1, 2009.

[3] R. Shen, S. Bubeck, S. Gunasekar, and A. Kumar, "Understanding neural network training from the perspective of feature learning," https://courses.cs.washington.edu/courses/cse543/23wi/schedule/lecture18.pdf, 2023, accessed: 2024-11-25.

[4] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[5] A. Name, "The ultimate beginner's guide to tensorflow," *Towards Data Science*, 2023, accessed: YYYY-MM-DD. [Online]. Available: https://towardsdatascience.com/the-ultimate-beginners-guide-to-tensorflow-41d3b8e7d0c7

[6] C. Szegedy, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

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

[8] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, 2017, pp. 506–519.

[9] A. Mądry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *stat*, vol. 1050, no. 9, 2017.

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

[11] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *Artificial intelligence safety and security*. Chapman and Hall/CRC, 2018, pp. 99–112.

[12] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, and D. Mukhopadhyay, "A survey on adversarial attacks and defences," *CAAI Transactions on Intelligence Technology*, vol. 6, no. 1, pp. 25–45, 2021.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[14] ——, "Identity mappings in deep residual networks," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*. Springer, 2016, pp. 630–645.

[15] C. Wang, "Federated learning with resnet-18 for medical image diagnosis," in *Proceedings of the 2023 8th International Conference on Multimedia Systems and Signal Processing*, 2023, pp. 34–39.

[16] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[17] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE signal processing magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[18] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, "Advances and open problems in federated learning," *Foundations and trends® in machine learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[19] E. Wong and Z. Kolter, "Provable defenses against adversarial examples via the convex outer adversarial polytope," in *International conference on machine learning*. PMLR, 2018, pp. 5286–5295.

[20] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1175–1191.

[21] M. Mohri, G. Sivek, and A. T. Suresh, "Agnostic federated learning," in *International conference on machine learning*. PMLR, 2019, pp. 4615–4625.

[22] W. Zhao, S. Alwidian, and Q. H. Mahmoud, "Adversarial training methods for deep learning: A systematic review," *Algorithms*, vol. 15, no. 8, p. 283, 2022.

[23] W. Brendel, J. Rauber, A. Kurakin, N. Papernot, B. Veliqi, S. P. Mohanty, F. Laurent, M. Salathé, M. Bethge, Y. Yu *et al.*, "Adversarial vision challenge," in *The NeurIPS'18 Competition: From Machine Learning to Intelligent Conversations*. Springer, 2020, pp. 129–153.

[24] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," *stat*, 2009.

[25] F. Chollet, *Deep learning with Python*. Simon and Schuster, 2021.

[26] S. MadhanMohan and E. Karthikeyan, "Classification of image using deep neural networks and softmax classifier with cifar datasets," in *2022 6th International Conference on Intelligent Computing and Control Systems (ICICCS)*. IEEE, 2022, pp. 1132–1135.

[27] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[28] H. G. L. Z. W. KQ, "Densely connected convolutional networks," 2018.

[29] C. M. Bishop, "Pattern recognition and machine learning," *Springer google schola*, vol. 2, pp. 1122–1128, 2006.

[30] H. Zhang, Y. Yu, J. Jiao, E. Xing, L. El Ghaoui, and M. Jordan, "Theoretically principled trade-off between robustness and accuracy,"

in *International conference on machine learning*. PMLR, 2019, pp. 7472–7482.

[31] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *International conference on artificial intelligence and statistics*. PMLR, 2020, pp. 2938–2948.

[32] A. Madry, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[33] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," *arXiv preprint arXiv:1611.01236*, 2016.