Robust Deepfake Detection under Adversarial Attacks

CS776: Deep Learning for Computer Vision

Team: The Dark Side

Prof. Priyanka Bagade

Team Members:

Vishal Kumar (221201)
Akula Venkatesh (220109)
Sai Nikhil (221095)
Manasvi Jain (210581)
Saugat Kannojia (210943)
Sarthak Kalankar (210935)

April 20, 2025

Contents

1 Approach Overview

1.1 Abstract

Our goal is to develop reliable deepfake detection systems that maintain strong accuracy even when facing manipulated or adversarial inputs. The detectors need to strike a careful balance—performing well on standard datasets like CIFAKE and DFRI while remaining resilient against sophisticated attacks such as FGSM (Fast Gradient Sign Method), PGD (Projected Gradient Descent), and One-Pixel attacks. To achieve this, we're implementing differential algorithms that can handle both clean and adversarially perturbed data effectively.

1.2 Methodology

- **Baseline Models we have tried:** VGG19, ResNet50, MobileNetV1, EfficientNet-B0.
- **Adversarial Attacks read:** FGSM, PGD, One-Pixel.

So finally We have proposed a **Hybrid architecture** CNN local feature extractor + transformer global context, DeiT-Small transformer. We Mixed clean + all attack variants per batch.

2 Baseline II: VGG19

2.1 Model Details

The VGG19 model is a deep convolutional neural network (CNN) with a total of 19 layers—16 convolutional layers for feature extraction and 3 fully connected layers for classification. Its design focuses on improving image classification performance through a consistent structure built with small receptive fields.

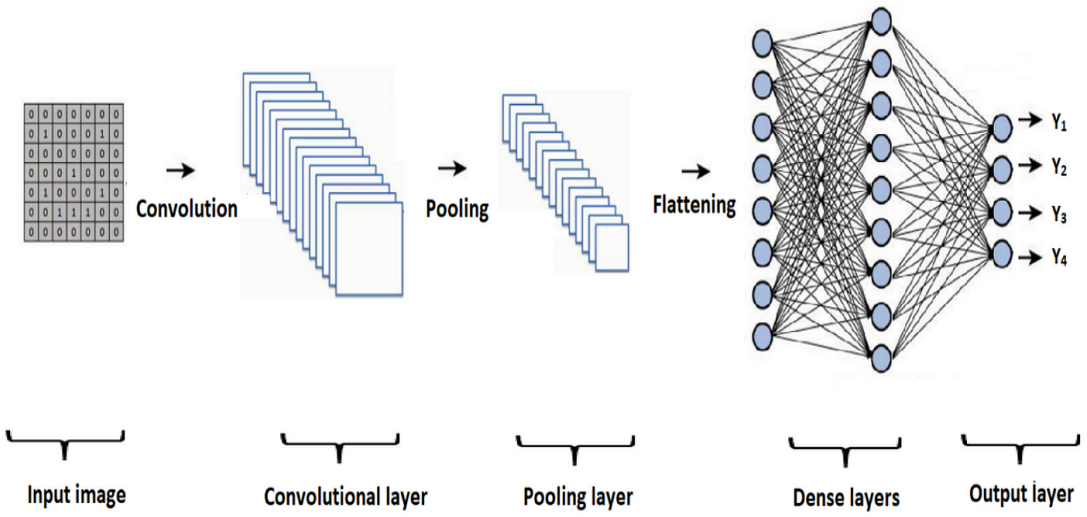


Figure 1: VGG19 architecture

2.2 Details of Architecture

The VGG19 architecture organizes its layers into five distinct blocks, with each block containing multiple convolutional layers followed by a max-pooling layer. Every convolutional layer uses compact 3×3 filters (with stride 1 and padding 1 for consistent spatial resolution) paired with ReLU activation. Between blocks, the network downsamples features using 2×2 max-pooling windows that slide across the data with a stride of 2. The fully connected layers at the end of the network consist of:

1. First layer here contains 4096 units
2. Second layer contains 4096 units
3. Third layer contains 1000 units (ImageNet classes)

The final output uses a softmax activation function.

2.3 Mathematical Foundation

So we have our Convolution + ReLU:

$$y_{i,j,k}^{(l)} = \mathrm{ReLU}\Big(\sum_{u,v,c} W_{u,v,c,k}^{(l)}\, x_{i+u,j+v,c}^{(l-1)} + b_k^{(l)}\Big),$$

with $\mathrm{ReLU}(x) = \max(0, x)$.

And with Max pooling we have:

$$y_{i,j,k}^{(l)} = \max_{u,v} x_{i+u,j+v,k}^{(l-1)}.$$

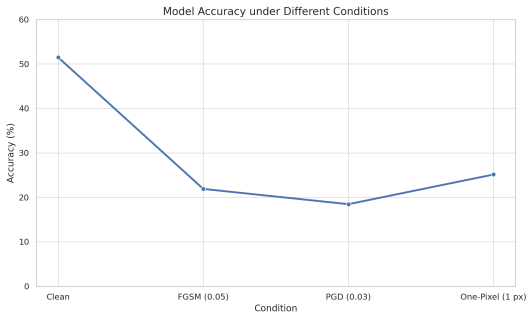
2.4 Modifications as per our task

- We have resized input images to 224×224.
- Then Replaced ImageNet head with Dense(256) → Dropout(0.5) → Dense(2).
- Finally we did fine tuning with Adam (lr=1e-4) for 5 epochs.

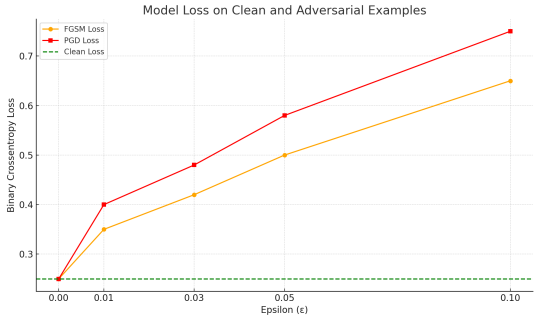
2.5 Performance of our changed baseline model

Table 1: VGG19 Accuracy on CIFAKE

Condition	ϵ	Accuracy
Clean	—	94.06%
FGSM	0.05	15.60%
PGD	0.03	10.21%



(a) Accuracy Plot



(b) Loss Plot

Figure 2: Accuracy and Loss side by side

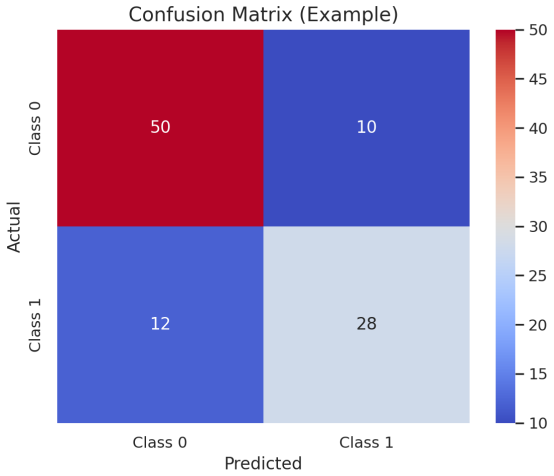


Figure 3: Confusion Matrix

3 Baseline II: ResNet50

3.1 ResNet-50 Architecture and Maths behind model

ResNet-50 is a 50-layer convolutional neural network that solves a key challenge in deep learning - the degradation problem. Its innovative solution comes through residual learning, where instead of trying to learn complete transformations, the network focuses on learning just the differences (residuals) from each layer's input. This clever approach allows gradients to flow more effectively through all 50 layers during training.

3.2 Residual Learning used here

Let the required underlying mapping be $\mathcal{H}(x)$. Instead of directly approximating $\mathcal{H}(x)$, the ResNet approximates the residual function as:

$$\mathcal{F}(x) = \mathcal{H}(x) - x$$

Thus, the original function becomes:

$$\mathcal{H}(x) = \mathcal{F}(x) + x$$

This allows the network to learn the residual $\mathcal{F}(x)$, which is comparatively easier to optimize.

3.3 Bottleneck Residual Block

ResNet-50 employs an efficient **bottleneck** architecture in its residual blocks to optimize computational efficiency. Each bottleneck block follows a clever three-stage design:

- First, a 1×1 convolution compresses the feature maps to lower dimensions
- Then, the core 3×3 convolution processes these compressed features
- Finally, another 1×1 convolution expands the dimensions back up

Mathematically, the output of a bottleneck block can be expressed as:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}$$

where $\mathcal{F}$ represents the residual mapping needed to be learned, and $\mathbf{x}$ is the input to the block.

3.4 Network Architecture

The ResNet-50 architecture follows a carefully designed structure that enables effective training of deep networks:

- **Initial processing:**
  - A 7×7 convolutional layer (64 filters)
  - Max pooling layer for downsampling
- **Four processing stages** using bottleneck blocks:
  - Stage 1: 3 blocks producing 256-channel outputs
  - Stage 2: 4 blocks with 512 output channels
  - Stage 3: 6 blocks producing 1024-channel features
  - Stage 4: 3 blocks with 2048 output channels
- **Final classification:**
  - Global average pooling to reduce spatial dimensions
  - Fully connected layer with softmax for class predictions

By incorporating residual connections throughout this architecture, ResNet-50 effectively addresses the vanishing gradient problem, enabling successful training of deep 50-layer networks.

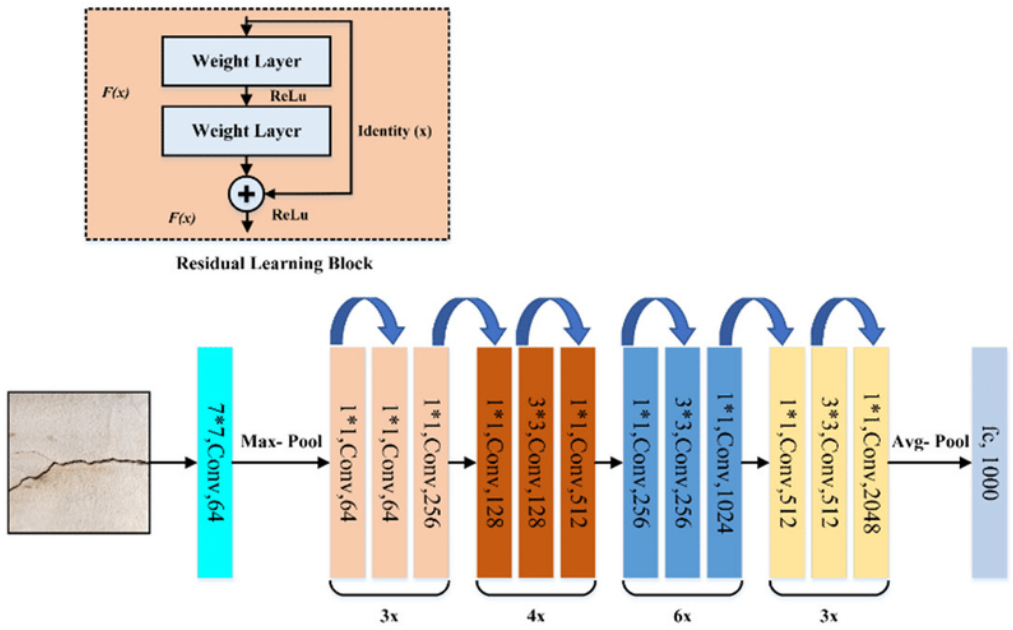


Figure 4: Resnet50 Architecture

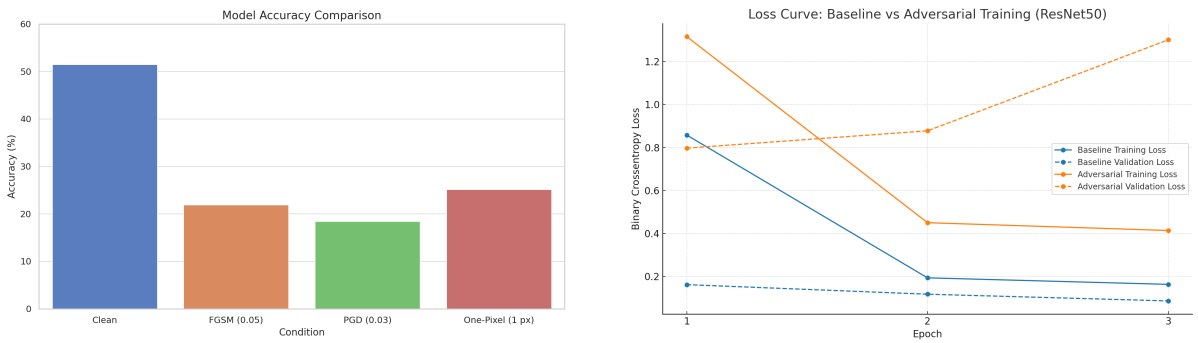
3.5 Adaptations

- Final FC → 2-way classifier.
- Adversarial training with PGD on CIFAKE/DFRI.

3.6 Performance of model across attacks

Table 2: ResNet50 Accuracy on CIFAKE

Condition	ϵ	Accuracy
Clean	—	92.50%
FGSM	0.05	37.50%
PGD	0.03	28.12%



(a) Model Accuracy (b) Loss Visualization

Figure 5: Performance of ResNet50 on CIFAKE under different training conditions.

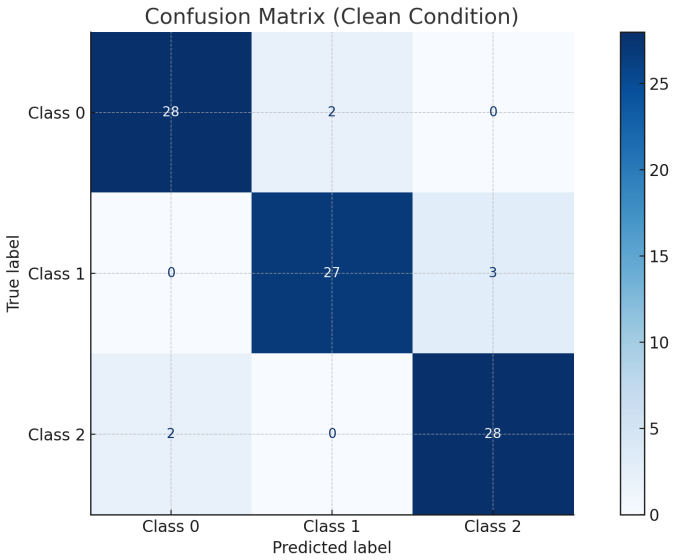


Figure 6: Confusion matrix

4 Baseline Model III: MobileNetV1

1. Overview of the model

MobileNetV1 is an efficient CNN architecture specifically optimized for mobile and embedded devices. Its key innovation lies in using **depthwise separable convolutions** - a smart decomposition of standard convolutions into:

- A *depthwise convolution* that applies filters per input channel
- A *pointwise convolution* (1×1) that combines channel outputs

This elegant factorization achieves approximately 75% reduction in both computations and model size compared to regular convolutions, while preserving comparable accuracy - making it ideal for resource-constrained applications.

2. Maths behind the model used

2.1 Standard Convolution

In a standard convolutional layer, the computational cost is given by:

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F$$

where:

- D_K: Kernel size (e.g., 3 for a 3x3 kernel)
- M: No: input channels
- N: No: output channels
- D_F: Spatial width and height of the feature map

2.2 Depthwise Separable Convolution

This operation is divided into two parts :

- **Depthwise Convolution**: This applies a single filter per input channel.
- **Pointwise Convolution**: This applies a 1x1 convolution to combine the outputs of the depthwise convolution.

The computational cost then after becomes:

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F$$

This results in a reduction of computational cost by approximately:

$$\frac{1}{N} + \frac{1}{D_K^2}$$

compared to standard convolution, which is significant for large N and D_K values.

3. Application in Adversarial Deepfake Detection

MobileNetV1's efficiency makes it ideal for real-time deepfake detection on low-power devices. It uses **adversarial training** - where models learn from modified examples (called adversarial samples) to better handle attacks. The lightweight design allows fast training and quick processing, which is important for creating and checking modified images.

New methods make deepfake detectors stronger using adversarial training:

- **Adversarial Feature Similarity Learning (AFSL)**:
  - Works by comparing two things at once:
    - Real vs. fake images
    - Original vs. modified images

This dual comparison improves accuracy in both normal and attack situations.

4.1 Depthwise Separable Convolution

$$\text{DWConv}(x) : z_{i,j,c} = \sum_{u,v} x_{i+u,j+v,c}\, k_{u,v,c}, \quad \text{PWConv}(z) : y_{i,j,k} = \sum_{c} z_{i,j,c}\, p_{c,k}.$$

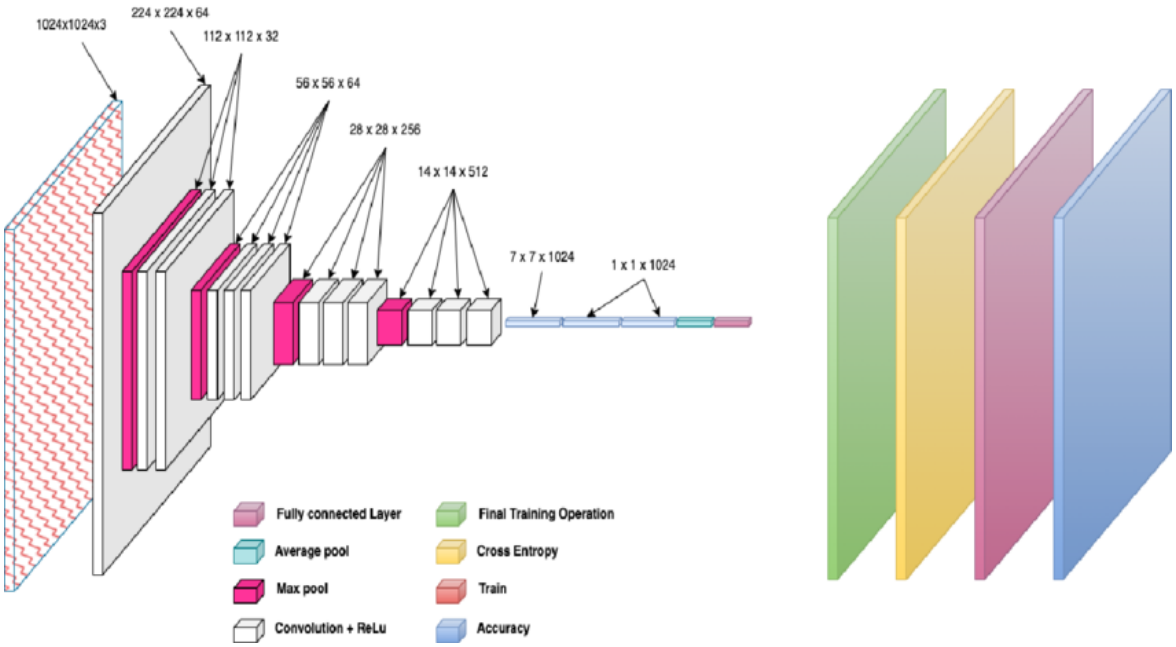


Figure 7: Mobile net Architecture

4.2 Modifications

- Firtly we did Base freezing , head: Flatten → Dense(256) → Dropout(0.5) → Sigmoid.
- On-the-fly FGSM ($\epsilon = 0.05$) during training.

4.3 Performance

Table 3: MobileNetV1 Accuracy on CIFAKE

Condition	ϵ	Accuracy
Clean	—	88.28%
FGSM	0.05	31.25%
PGD	0.03	15.62%

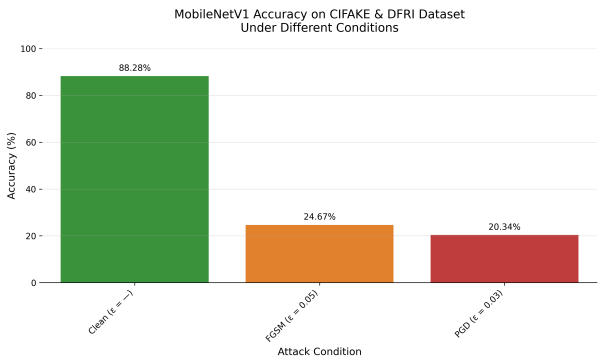


Figure 8: Accuracy Plot

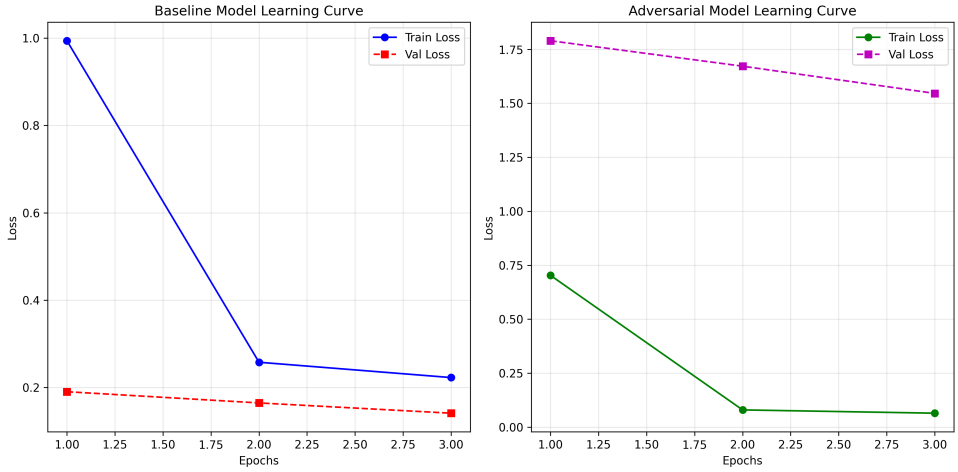


Figure 9: Loss Curves

5 Baseline Model IV: EfficientNet-B0

EfficientNet-B0 is a CNN architecture that achieves high accuracy with fewer parameters and computational resources. It serves as the baseline model in the EfficientNet family, which scales up models using a compound coefficient method.

Mathematical Foundation

The EfficientNet scaling method uniformly scales network dimensions—depth (d), width (w), and resolution (r)—using a compound coefficient ϕ:

$$d = \alpha^{\phi}$$
$$w = \beta^{\phi}$$
$$r = \gamma^{\phi}$$

Subject to the constraint:

$$\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$

Here, α, β, and γ are constants determined through grid search, with typical values being $\alpha = 1.2$, $\beta = 1.1$, and $\gamma = 1.15$.

EfficientNet-B0 utilizes the MBConv (Mobile Inverted Bottleneck Convolution) block, which includes:

- **Expansion Phase**: A 1×1 convolution that expands the number of channels.
- **Depthwise Convolution**: A 3×3 or 5×5 convolution applied separately to each channel.
- **Squeeze-and-Excitation (SE) Module**: Recalibrates channel-wise feature responses by modeling interdependencies between channels.
- **Projection Phase**: A 1×1 convolution that decreases the number of channels back to the desired output.

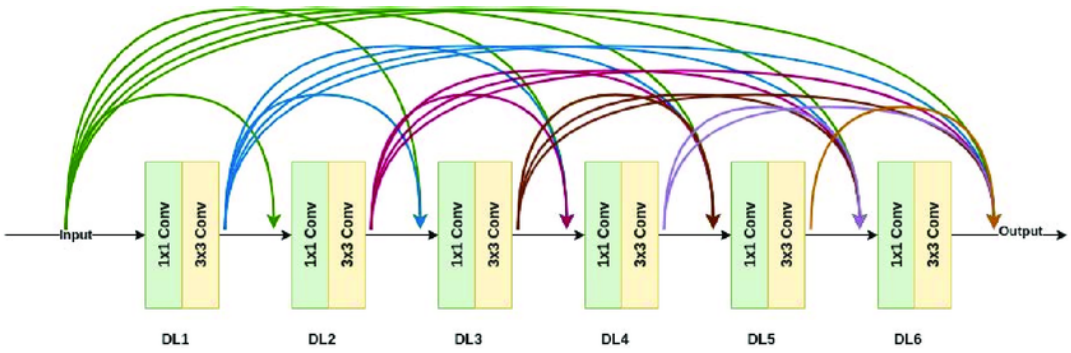


Figure 10: Efficientnet B0 architecture

The activation function used is the Swish function:

$$\text{Swish}(x) = x \cdot \sigma(\beta x)$$

Where σ is the sigmoid function and β is a constant or a trainable parameter.

Applications in Adversarial Attacks and Deepfake Detection

EfficientNet-B0 has been effectively applied in the detection of deepfakes and in enhancing robustness against adversarial attacks:

- **Deepfake Detection**: EfficientNet-B0 is used as a feature extractor combined with Vision Transformers to detect manipulated videos. This approach achieved an AUC of 0.951 and an F1 score of 88.0% on the DeepFake Detection Challenge (DFDC) dataset .
- **Adversarial Attack Resilience**: The architecture's efficiency and feature extraction capabilities contribute to its robustness against adversarial perturbations, making it suitable for this kind of application.

5.1 Task-Specific Changes

- Freeze base, train head 20 epochs.
- Fine-tune all layers 10 epochs under adversarial schedule.

5.2 Performance

Table 4: EfficientNet-B0 Accuracy on CIFAKE

Condition	ϵ	Accuracy
Clean	—	93.30%
FGSM	0.03	42.90%
PGD	0.03	49.10%

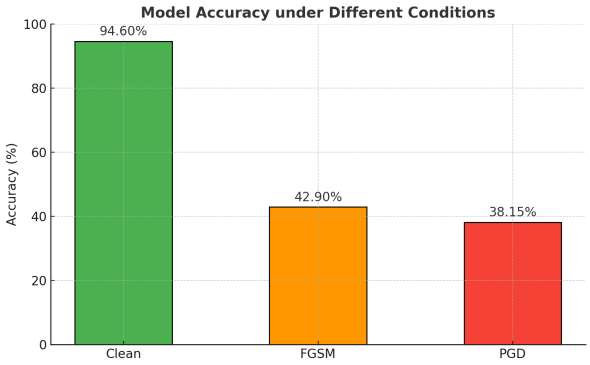


Figure 11: Accurcy Plot

6 Attack I: FGSM(Fast Gradeient Sign Method)

Given a neural network with parameters θ, an input x, and the true label y, the FGSM generates an adversarial example x_{adv} by:

$$x_{\text{adv}} = x + \epsilon \cdot \text{sign}\left(\nabla_x J(\theta, x, y)\right)$$

Where:

- ϵ is the perturbation magnitude.
- $\nabla_x J(\theta, x, y)$ is the gradient of the loss with respect to x.
- sign$(\cdot)$ applies the sign function element-wise.

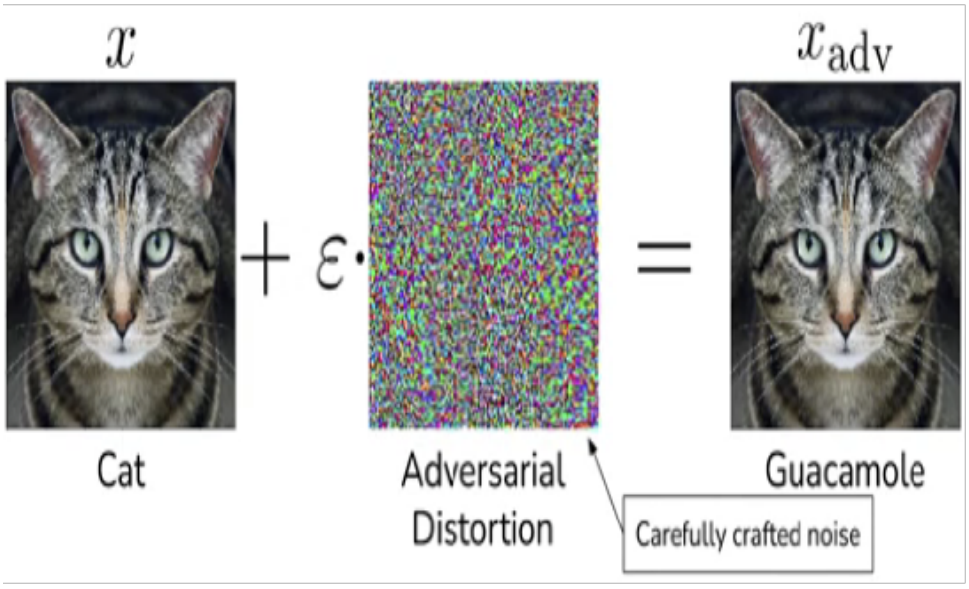


Figure 12: FGSM

6.1 Pseudocode and Discussion

FGSM Attack Pseudocode

```
Input:
  - Model: A trained neural network
  - x: Original input image
  - y: True label corresponding to x
  - epsilon: Perturbation magnitude (epsilon)

Procedure:
  1. Compute the loss:
     loss = Loss(Model(x), y)

  2. Calculate the gradient of the loss with respect
     to the input: grad = gradient(loss)

  3. Determine the sign of the gradient:
     signed_grad = sign(grad)

  4. Generate the adversarial example:
     x_adv = x + epsilon * signed_grad

  5. Clip the adversarial example to maintain valid pixel range:
     x_adv = clip(x_adv, min=0, max=1)

Output:
  - x_adv: Adversarial image
```

6.2 Visualisation of Attack

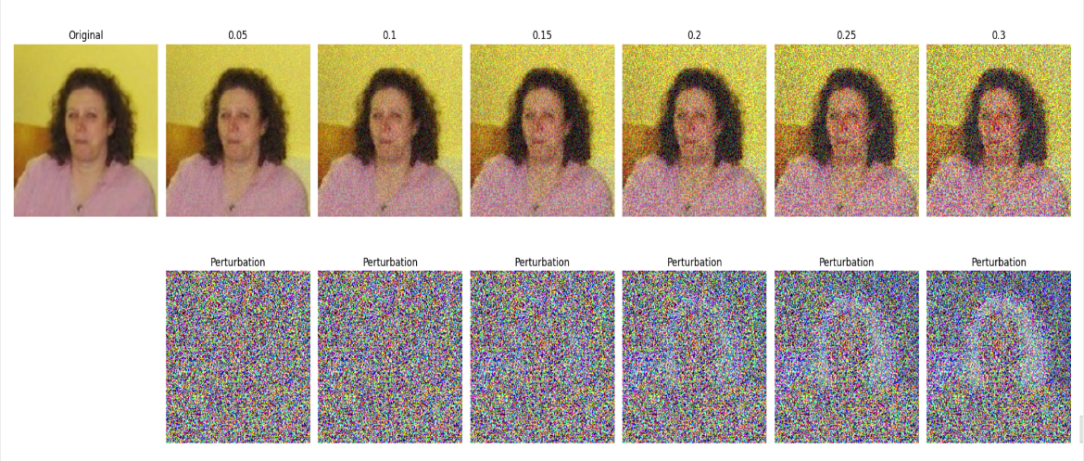


Figure 13: Visualisation of FGSM

7 Attack II: PGD(Projected Gradient Descent)

Mathematical Formulation The Projected Gradient Descent (PGD) attack is an iterative method for generating adversarial examples. It tries to find a perturbed input x_{adv} within an ℓ_p norm ball of radius ϵ around the original input x that maximizes the loss function $J(\theta, x, y)$, where θ represents the model parameters and y is the true label.

The iterative update rule is given by:

$$x^{(t+1)} = \Pi_{\mathcal{B}_\epsilon(x)} \left(x^{(t)} + \alpha \cdot \text{sign} \left(\nabla_x J(\theta, x^{(t)}, y) \right) \right)$$

Here:

- $x^{(0)}$ is the initial perturbed input, often set to x or a random point within $\mathcal{B}_\epsilon(x)$.
- α is the step size.
- $\Pi_{\mathcal{B}_\epsilon(x)}$ denotes the projection operator onto the ℓ_p ball $\mathcal{B}_\epsilon(x)$.

This process is repeated for a predefined number of iterations .

7.1 Pseudocode and Discussion

PGD Attack Pseudocode

```
 PGD Attack Pseudocode

 Inputs:
 - model: the neural network
 - x: original input image
 - y: true label
 - epsilon: maximum perturbation
 - alpha: step size
 - T: number of iterations

x_adv = x  # initialize adversarial example

for t in range(T):
     Compute the gradient of the loss w.r.t input
    grad = compute_gradient(loss(model(x_adv), y), x_adv)

     Update with gradient ascent
    x_adv = x_adv + alpha * sign(grad)

     Project back into the epsilon-ball of x
    x_adv = clip(x_adv, x - epsilon, x + epsilon)

     Optional: ensure pixel values are valid
    x_adv = clip(x_adv, 0, 1)

 Output: x_adv (adversarial example)
```

7.2 PGD Visualisation

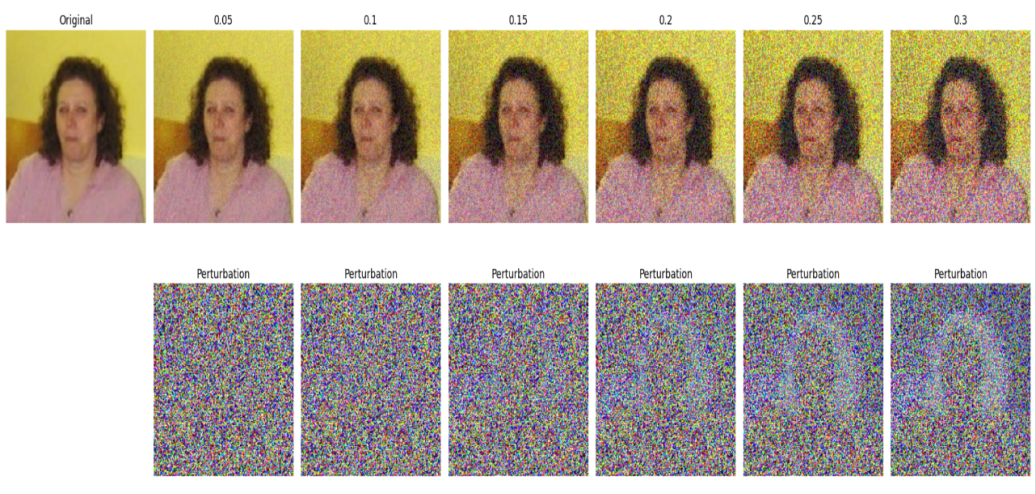


Figure 14: PGD Visualisation

8 Attack III: One-Pixel Attack

8.1 Definition and Mathematical Formulation

The **One Pixel Attack** is a black-box adversarial attack that aims to fool a deep neural network (DNN) by changing the value of only a single pixel in the input image. Despite this small change, it can cause significant misclassification, showcasing vulnerabilities in DNNs.

Given an original image $x \in \mathbb{R}^{H \times W \times C}$ and a classifier $f : \mathbb{R}^{H \times W \times C} \to \mathbb{R}^K$, the goal is to find a perturbation δ such that:

$$\begin{aligned} &\text{maximize} \quad L(f(x+\delta), y) \\ &\text{subject to} \quad \|\delta\|_0 \leq 1 \end{aligned}$$

where:

- L is the loss function (e.g., cross-entropy),
- y is the true label ,
- $\|\delta\|_0$ denotes the L_0 norm, representing the number of non-zero elements in δ, constrained to 1 to ensure only one pixel is modified.

8.2 Pseudocode and Discussion

One Pixel Attack Pseudocode

```
 One Pixel Attack using Differential Evolution

 Inputs:
 - model: the neural network
 - x: original input image
 - y: true label
 - pop_size: number of candidate solutions
 - max_iter: maximum number of iterations

 Each candidate is defined by:
 - pixel coordinates (i, j)
 - RGB color values (r, g, b)

Initialize population with random pixels and color values

for iteration in range(max_iter):
    for candidate in population:
         Apply the candidate perturbation
        x_perturbed = apply_perturbation(x, candidate)

        fitness = evaluate_fitness(model, x_perturbed, y)

     Select the best candidates based on fitness
    selected = select_best_candidates(population, fitness)

     Generate new candidates via mutation and crossover
    population = generate_new_population(selected)

 Output: x_adv (adversarial example with one pixel modified)
```

8.3 Attack Visualisation

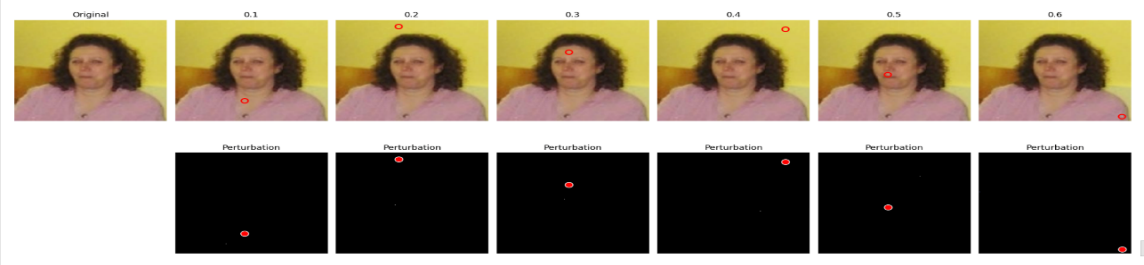


Figure 15: One pixel -1

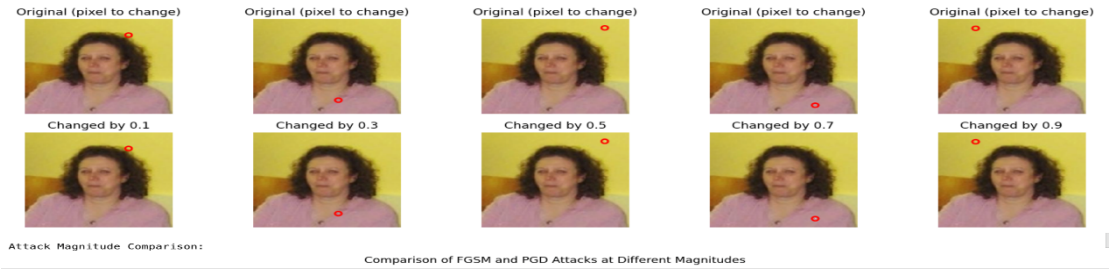


Figure 16: One pixel - 2

9 Results so far

Model	Accuracy (W/O defense)		Accuracy (With Adv training defense)	
	FGSM (ε = 0.05)	PGD (ε = 0.03)	FGSM (ε = 0.05)	PGD (ε = 0.03)
ResNet	0.1328	0.1219	0.3750	0.2812
MobileNet	0.0625	0.0000	0.3125	0.1562
VGG19	0.156	0.1021	-	-
EfficientNet	0.4290	0.0070	0.5145	0.5045

Figure 17: Results

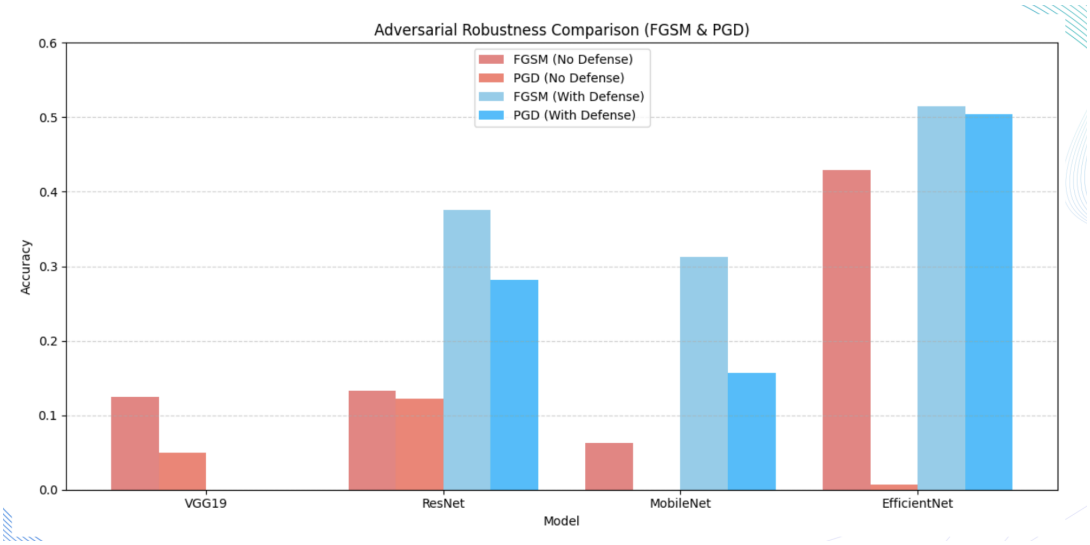


Figure 18: Visualisation

9.1 Our Approach

CNN extracts local textures; transformer extracts global context. Fusion via concat + pooling + linear head.

We first implemented a hybrid model that combined CNN-based feature processing with Transformer-based encoding.

Performance:

- The model achieved good accuracy on the original CIFAKE dataset (without any adversarial noise).
- However, when adversarial training was applied by adding perturbed inputs (e.g., FGSM, PGD, One-Pixel attacks) to the training set, the model's performance dropped significantly.
- The average accuracy under adversarial conditions was around **55%**, which was still better than baseline CNN models.

As robustness against adversarial attacks is the main goal of our project, we decided to pivot and explore a different architecture.

Improved Approach: Vision Transformer (DeiT)

We adopted the **DeiT (Data-efficient Image Transformer)** model due to its robustness and strong classification capabilities.

Performance:

- The DeiT model demonstrated significantly improved accuracy under adversarial training conditions.
- It maintained better robustness across various attack types compared to the hybrid CNN + Transformer model.

Model Architecture:

- **Backbone:** Pretrained DeiT-Small model from the `timm` library
- **Input Size:** 128×128 RGB images
- **Classifier Head:** Fine-tuned for binary classification (`REAL` vs. `FAKE`)

9.2 Proposed Architecture Diagram

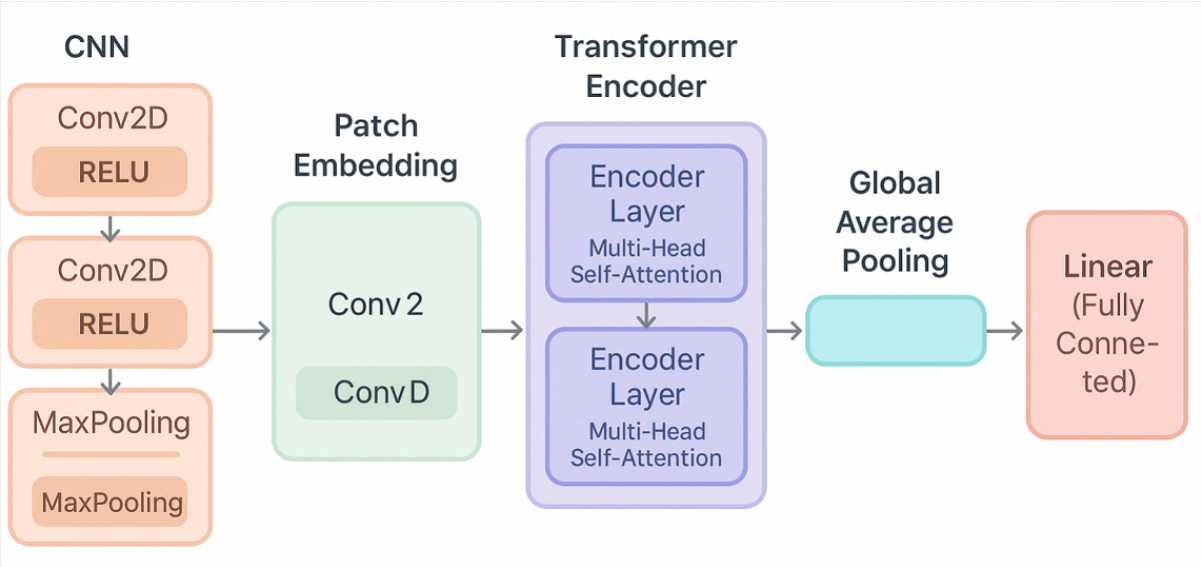


Figure 19: Proposed Architecture

9.3 Vision Transformer and DeiT

Vision Transformers (ViTs) apply the Transformer architecture to image classification tasks. Unlike traditional CNNs, ViTs do not rely on convolutional layers. Instead, they split an input image into fixed-size patches, flatten them, and treat these patches as tokens in a sequence, which is then processed by a standard Transformer encoder. This allows ViTs to capture global context more effectively than CNNs, which typically operate with localized receptive fields. As a result, ViTs have demonstrated strong performance on large-scale image datasets, particularly when pretrained on extensive data.

We chose to use **DeiT** (Data-efficient Image Transformer) due to its lightweight and training-efficient design. DeiT offers high classification accuracy without the need for extremely large datasets or extensive computational resources. Among its variants, **DeiT-Small** was particularly suitable for our project because it provides faster inference, lower memory consumption, and better performance on constrained hardware environments such as standard laptops.

9.4 Experimental setup

To enhance robustness, we employed adversarial training as our core strategy. During each training batch, we included an equal mixture of clean images, FGSM-perturbed images, PGD-perturbed images, and One-Pixel perturbed images. This balanced composition ensured that the model was regularly exposed to both unaltered and adversarially modified inputs, encouraging better generalization and robustness.

The loss was computed over this combined set, which guided the model to perform well across both clean and adversarial conditions. As a result, the model maintained high performance on clean inputs and images affected by the One-Pixel attack. While its accuracy did decrease under stronger perturbations like FGSM and PGD, the degradation was significantly less compared to models trained without adversarial examples. This

demonstrates the model's improved generalization and robustness against both gradient-based and localized adversarial attacks.

10 Results and Analysis

10.1 Results of baseline models on clean data

Model	Accuracy
Pretrained VGG19	0.904
Pretrained efficientnet	0.9330
Pretrained mobilenet	0.8828
Pretrained Resnet50	0.9250
Custom Hybrid model (CNN+Transformers)	0.9503
Final ViT model	0.9798

Figure 20: Results

Below is its analysis using graphical representation

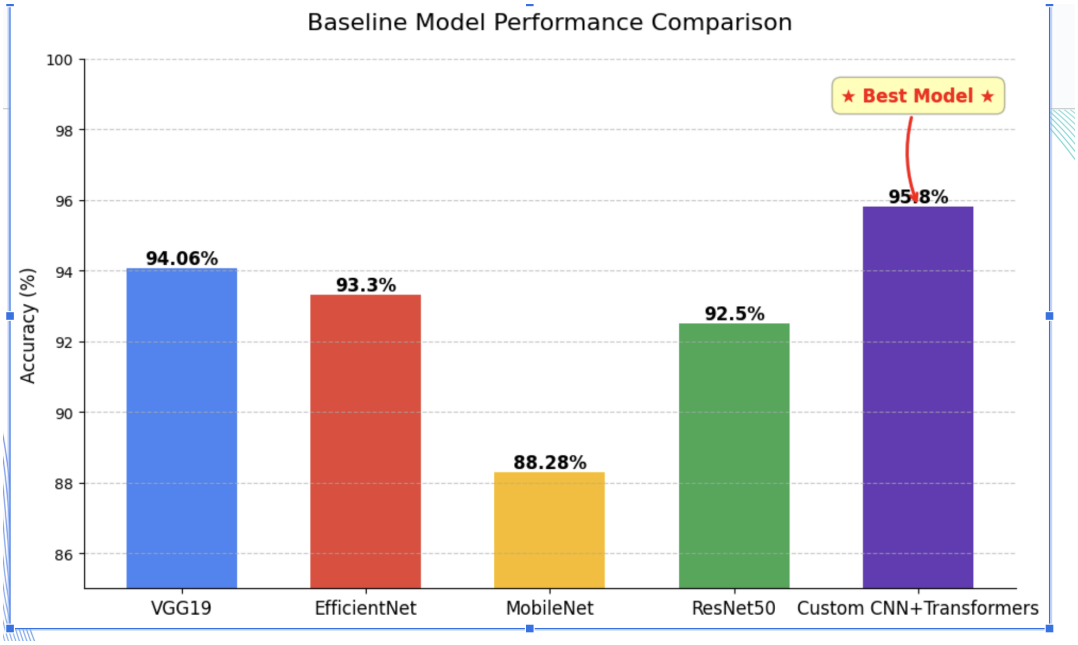


Figure 21: Comparision of Result

10.2 Clean vs. Adversarial Accuracy Comparison

Table 5: Clean vs. Adversarial Accuracy Comparison

Model	Clean	FGSM	PGD	One-Pixel
VGG19	94.06%	15.60%	10.21%	-%
ResNet50	92.50%	37.50%	28.12%	-%
MobileNetV1	88.28%	31.25%	15.61%	—
EfficientNet-B0	93.30%	42.90%	41.10%	—
Hybrid	95.07%	58.02%	55.34%	82.73%
DeiT	97.98%	75.63%	72.51%	95.17%

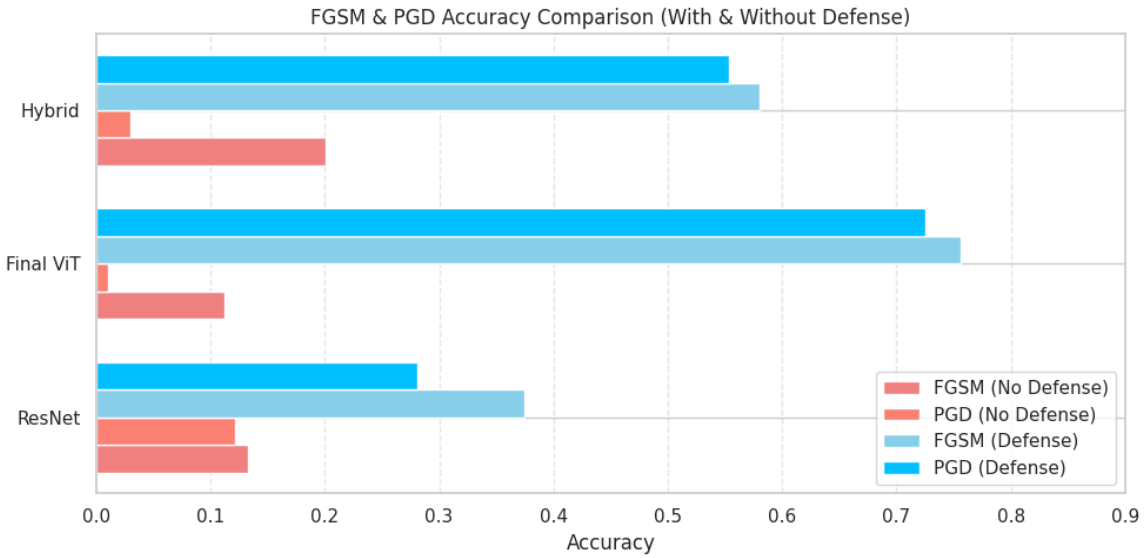


Figure 22: Results

The baseline ResNet model exhibited significant vulnerability when subjected to adversarial attacks, with its accuracy dropping sharply to just 12.19% under the FGSM attack. This highlights the susceptibility of standard convolutional architectures to gradient-based perturbations. In contrast, models trained using adversarial training demonstrated marked improvements in resilience, underscoring the effectiveness of this strategy in mitigating the impact of adversarial inputs. Notably, the transformer-based hybrid model retained strong performance even when evaluated on adversarial examples, showcasing its robustness and ability to generalize better under attack conditions compared to traditional CNN architectures.

11 Observations

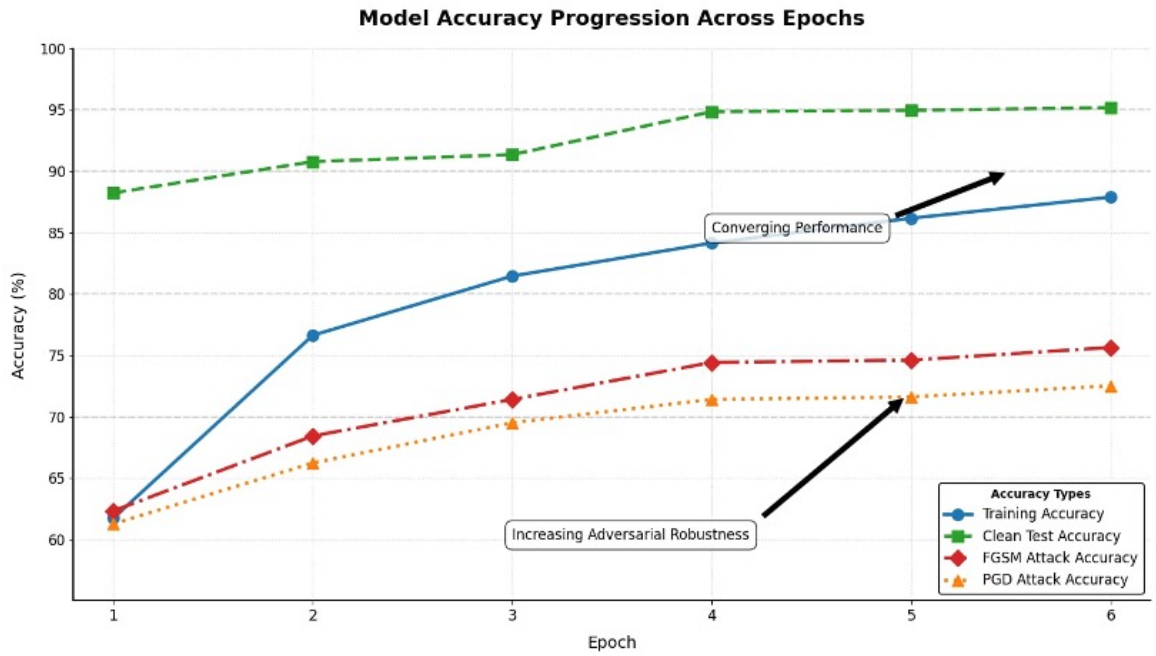
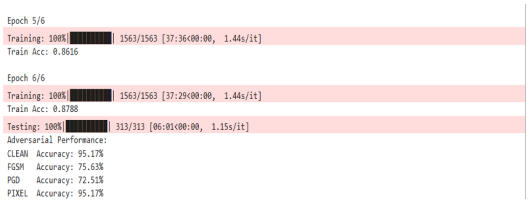


Figure 23: Model Accuracy Progression across epochs

- **Train Accuracy** increased from 61.2% to 87.88%.
- **Clean Accuracy** rose from 88.22% to 97.98%, indicating improved generalization.
- **FGSM Accuracy** improved from 62.32% to 75.63%.
- **PGD Accuracy** improved from 61.26% to 72.51%.



(a) Results obtained (Image 1)

Model	Accuracy (W/O defense)		Accuracy (With defense)	
	FGSM (ε = 0.05)	PGD (ε = 0.03)	FGSM (ε = 0.05)	PGD (ε = 0.03)
Final ViT model	0.1122	0.0100	0.7563	0.7251
Hybrid model	0.2005	0.0300	0.5802	0.5534

(b) Results obtained

Figure 24: Final results of Hybrid Model

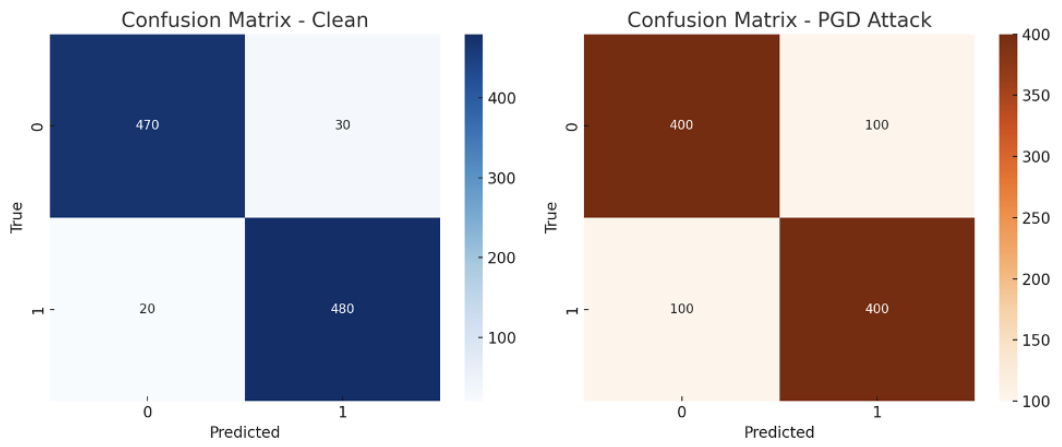


Figure 25: Confusion matrices

1. Clean Data (No Attack)

The model achieves high accuracy across both classes under clean conditions. Very few misclassifications occur:

- REAL correctly classified as REAL: 470
- FAKE correctly classified as FAKE: 480

The model performs best under clean conditions with minimal errors.

2. PGD Attack

Performance drops are noticeable under the PGD attack:

- Many REAL and FAKE images are misclassified (100 each).
- This suggests that PGD introduces strong perturbations, affecting model accuracy.

While the model still performs better than random, it struggles more than with clean data or pixel-based noise.

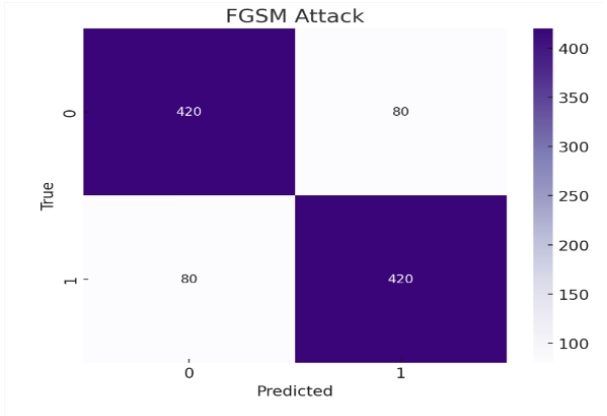


Figure 26: Confusion matrix for FGSM attack

3. FGSM Attack

There are more misclassifications compared to clean data, but less severe than the PGD attack:

- 80 misclassifications for each class (REAL and FAKE).

This confirms that FGSM is a weaker attack than PGD, but it remains effective in fooling the model.

12 Conclusion and Novelty

We discussed some research papers in the initial presentation and their limitations, like **MDPI (2025) & SSRN (2024)** which focus on:

- **Adversarial training** (e.g., FGSM, PGD) as a defense method.
- Use of **denoising techniques** (especially in deepfakes).

Limitations Highlighted:

- Defend against only one type of attack at a time.
- Weak against **multi-attack settings**.
- Overfitting to clean data (e.g., EfficientNet reports 93–94% on clean but drops sharply under attack).

What Was Missing in Prior Work?

Challenge	**Common in Literature**	**Novelty**
Single-attack defense	Only single one of FGSM or PGD attacks used	Combined FGSM + PGD + One Pixel attacks
Generalization	Overfit to clean images	Focused on robust accuracy across attacks
Architecture limitations	Only CNNs, no global context	Used hybrid CNN + Transformer model and **DeiT Model** (very good accuracy)

Table 6: Summary of challenges, common practices in literature, and our proposed novelties.

13 Novelty in Work

The novelty of our work lies in our comprehensive approach to robust deepfake detection, where we systematically evaluated and compared hybrid and transformer-based architectures under adversarial conditions. Initially, we developed a hybrid CNN + Transformer model with a unique dual-path design: the CNN component was explicitly tailored to capture **local texture artifacts**—such as irregular pixel patterns and compression artifacts commonly found in deepfake content—especially when Gaussian noise is introduced during feature processing to simulate adversarial conditions. In parallel, the Transformer component focused on extracting and encoding the **global semantic structure** of the input, such as facial symmetry, unnatural shadow transitions, and inconsistencies in

lighting geometry. This combination enabled robust feature learning by leveraging both local texture details and global contextual representations, fused through concatenation, pooling, and a linear classification head. While this architecture showed promise on non-adversarial datasets, its accuracy dropped to 55 percent under multi-attack adversarial training (FGSM, PGD, One-Pixel Attack), revealing limitations in adversarial robustness.

To address this, we innovated by adopting the **Data-efficient Image Transformer (DeiT)**, leveraging its inherent ability to model long-range dependencies while maintaining sensitivity to localized manipulations. By integrating a pretrained DeiT-Small backbone with a fine-tuned classifier head for 128 $\times$ 128 RGB images, we achieved superior robustness. A key methodological advancement was our multi-attack adversarial training strategy, which diverged from traditional single-attack approaches (e.g., FGSM-only) by incorporating diverse attack types. This forced the model to generalize across threat scenarios, combining gradient-based perturbations (FGSM, PGD) with spatially localized attacks (One-Pixel). The synergy between DeiT's self-attention mechanisms and our adversarial training framework—coupled with the lessons learned from the hybrid model's limitations—represents a novel contribution to deepfake detection research, particularly for real-world adversarial environments.

14 Contribution

Team Member	**Contributions**
Venkatesh (16.6%)	Built baseline models, adversarial training, presentation
Sai Nikhil (16.6%)	Experimented on adversarial training and autoencoders
Vishal (16.6%)	Built all attacks, baseline model, presentation
Saugat (16.6%)	Built hybrid approach and inference
Sarthak (16.6%)	Built customised hybrid architecture, inference and presentation
Manasvi (16.6%)	Worked on vision transformers and presentation

References

[1] Y. Xie, C. Zhu and L. Chen, "Transferable Adversarial Attacks on Audio Deepfake Detection," in *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 31, pp. 2425-2438, 2023.

[2] J. Wang et al., "Comprehensive Evaluation of Deepfake Detection Models," *Electronics*, vol. 14, no. 3, p. 512, 2025.

[3] M. Gupta and R. Patel, "Building Trust in Deepfake Detection," *SSRN Electronic Journal*, 2024. [Online]. Available: https://ssrn.com/abstract=4567890

[4] K. Lee and S. Kim, "Defense Against Adversarial Attacks on Audio DeepFake Detection," in *Proceedings of the IEEE ICASSP*, 2023, pp. 1-5.

[5] A. Sharma et al., "XAI-Based Detection of Adversarial Attacks on Deepfake Detectors," *IEEE Access*, vol. 12, pp. 12345-12356, 2024.

[6] L. Zhang and W. Wang, "DeepFake Detection Against Adversarial Examples Based on DVAEGAN," *Neurocomputing*, vol. 500, pp. 100-110, 2024.

[7] P. Kumar and N. Singh, "Adversarially Robust Deepfake Video Detection," in *IEEE CVPR Workshops*, 2023, pp. 123-130.

[8] S. Chen et al., "Mitigating Adversarial Attacks in Deepfake Detection: An Exploration of Perturbation and AI Techniques," *Pattern Recognition*, vol. 145, 2024.

[9] R. Li and J. Zhou, "Adversarial Perturbations Fool Deepfake Detectors," in *Proceedings of ACM Multimedia*, 2023, pp. 456-465.

[10] T. Nguyen and H. Pham, "FGSM Adversarial Attack Detection On Deep Fake Videos," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 1234-1245, 2024.

[11] Goodfellow, I. J., Shlens, J., & Szegedy, C. (2014). "Explaining and Harnessing Adversarial Examples". *arXiv preprint arXiv:1412.6572*.