DeepFake Detection

*Note: Sub-titles are not captured in Xplore and should not be use d

1st Nguyen Van Phong
FPT University
Hue, Viet Nam
phongnv.ce192081@gmail.com

2nd Dao Anh Khoa
FPT University.)
Dong Thap, Viet Nam
khoada.ce190399@gmail.com

3rd Huynh Anh Phuong
FPT University
An Giang, Viet Nam
haphuong12a8.2122@gmail.com

4th Tran Trung Nhan
FPT University)
Ca Mau, Viet Nam
baghet27@gmail.com

5th Huynh Ngoc Nhu Quynh
FPT University
Soc Trang, Viet Nam
quynhhuynh.thamestrinity@gmail.com

***Abstract*—This paper explores the use of deep learning models for detecting deepfakes, with a focus on leveraging the FaceForensics++ dataset for training and evaluation. We propose a comprehensive framework involving data preprocessing, augmentation, and the use of various neural network architectures, including feedforward neural networks, convolutional neural networks (CNN), residual networks (ResNet), and EfficientNet B0. Our study evaluates the performance of these models using key metrics such as accuracy, precision, and recall. We highlight the challenges of deepfake detection, such as overfitting and the continuous evolution of deepfake methods, and discuss potential solutions, including fine-tuning, optimization strategies, and the integration of advanced techniques like dropout and early stopping. The results demonstrate that the optimized CNN model, after fine-tuning, achieves an accuracy of 81.22%, significantly improving upon earlier versions. This research contributes to the real-world challenge of mitigating the risks posed by deepfake content, offering a practical solution to enhance digital media security and trustworthiness. By improving detection capabilities, our framework can be integrated into platforms to reduce the potential harm caused by malicious deepfakes in various sectors, including politics, media, and law enforcement.**

I. Introduction

Deepfake technology, leveraging artificial intelligence to create highly realistic fake videos, poses significant security, ethical, and misinformation risks. To combat this, sophisticated detection systems are crucial, relying heavily on Deep Learning and high-quality training data. This research is required because of the widespread misuse of deepfake videos, which can lead to misinformation, identity theft, political manipulation, and cyberbullying. Deepfake detection systems are used in various fields, including media verification, digital forensics, and social media platforms, ensuring the integrity and authenticity of digital content.

FaceForensics++, a vital dataset containing deepfake videos generated using techniques like FaceSwap, DeepFake, and NeuralTextures, enables researchers to enhance deepfake detection models. Utilizing such publicly available datasets accelerates research and establishes benchmarks for evaluating algorithms. Researchers, however, face challenges like inconsistent data quality, legal and ethical considerations, and the rapid evolution of deepfake methods.

As deepfake technology advances, the integration of Deep Learning with datasets like FaceForensics++ is essential for developing effective detection systems, preserving digital content authenticity and fostering a secure digital environment. However, the continuous evolution of deepfake methods and ethical concerns surrounding privacy and consent are disadvantages. This research aims to contribute to the development of more robust detection algorithms to mitigate these risks.

II. Related Works

Recent advancements in deep learning have led to the development of various neural network architectures tailored for specific tasks such as image and video recognition. Convolutional Neural Networks (CNNs) [1] have become the foundational architecture for many computer vision tasks due to their ability to capture spatial hierarchies in images. Following CNNs, deeper architectures like ResNet [2] introduced skip connections to address the vanishing gradient problem, allowing the training of very deep networks with improved accuracy.

More recently, EfficientNet [3] introduced a novel scaling method that uniformly adjusts depth, width, and resolution, achieving state-of-the-art performance with fewer parameters. This approach has shown significant improvements in both accuracy and computational efficiency, making it particularly suitable for real-time deployment.

While these models have advanced the field, they still face challenges in balancing computational cost with accuracy, especially for resource-constrained environments.

Our work aims to build upon these foundations by exploring novel ways to optimize model performance while maintaining efficiency in real-time applications.

A. Data Collection

We utilize the FaceForensics++ dataset, which provides a comprehensive collection of deepfake videos generated using various manipulation techniques.

B. Data Augmentation

To enhance the training process, we apply various data augmentation techniques, such as rotation, scaling, and color adjustments, to increase the variability and robustness of the dataset.

C. Building the Model

- **Feedforward Neural Network:** We start with a basic shallow feedforward neural network to establish a baseline performance.
- **Convolutional Neural Network (CNN):** Given its strength in image recognition, we implement CNN architectures to capture spatial hierarchies in the video frames.
- **ResNet (Residual Network):** To further improve accuracy, we use ResNet, which allows us to train deeper networks by mitigating the vanishing gradient problem through skip connections.
- **EfficientNet B0:** To balance accuracy and performance for real-time deployment, we use EfficientNet B0. This architecture scales depth, width, and resolution uniformly, optimizing both model efficiency and accuracy, making it ideal for resource-constrained environments.

D. Fine-Tuning and Evaluation

We assess the performance of our models using various metrics, such as accuracy, precision, recall, and F1-score. Additionally, we conduct ablation studies to understand the impact of different components and configurations on the overall system performance.

E. Deploying on a Website Using Python Flask

To make the model accessible for real-world applications, we deploy it on a website using Flask, a lightweight Python web framework. This involves setting up a Flask application, defining routes for the main web page and upload functionality, and integrating the trained deepfake detection model into the application to process user inputs efficiently. Additionally, we design a simple and user-friendly HTML interface that enables users to upload files and view the detection results.

By structuring our study in this manner, we aim to create a comprehensive framework for deepfake detection, leveraging the strengths of various neural network architectures and rigorous evaluation methodologies.

III. Dataset Selection and Preprocessing for Deepfake Detection

A. Selecting the Dataset and Understanding Its Structure: FaceForensics++

FaceForensics++ is a dataset comprising 200 video sequences, with an equal distribution of real and fake videos. Each folder, "real" and "fake," contains 20 videos ranging from 20 seconds to 60 seconds in length. The dataset provides high-quality Full HD (1080p) resolution videos, making it suitable for training and evaluating deepfake detection models. This structure ensures balanced and diverse training data, facilitating effective model learning.

B. Data Preprocessing Steps

1) Extracting Video Frames: Since deepfake detection relies heavily on image-based analysis, we need to extract frames from videos. This process involves converting videos into individual frames while maintaining a consistent frame rate to ensure that our dataset captures sufficient variations in facial expressions and movements. We use OpenCV, a powerful computer vision library, to extract frames from the videos efficiently.

Fig. 1: Original video frame

2) Cropping Faces: To enhance the deepfake detection process, the next step is face cropping. This step focuses on isolating the face region from each frame, enabling the model to concentrate on critical features such as eyes, mouth, and other facial landmarks.

To refine the face cropping process for real frames, we use both MTCNN and MediaPipe for face detection. Frames are parsed through these tools to identify faces, and the Intersection over Union (IoU) is calculated to measure the overlap between the detected bounding boxes. Based on the MTCNN bounding box, a new bounding box (square) is defined where the edge length equals 1.25 times the longer edge of the original MTCNN bounding box (rectangle).

To process fake frames, we use MediaPipe for face detection. Each frame is parsed through MediaPipe's face recognition system to accurately locate the face region. A new bounding box (square) is then defined, where the edge length is set to 1.25 times the longer edge of the bounding box detected by MediaPipe (rectangle).

3) Resizing Images: To standardize the input dimensions for the deep learning model, the extracted frames are resized to ensure uniform resolution. This process optimizes computational efficiency and ensures consistency in feature extraction. Specifically, the images are resized to dimensions of 224x224 pixels with three color channels (RGB format), maintaining the integrity of their original color information.

Fig. 2: Original face-cropped part

4) Data Augmentation: To enhance the variability of our training dataset, a comprehensive data augmentation process is applied. This includes techniques such as Gaussian blur, Gaussian noise, brightness and contrast adjustments, hue and saturation shifts, small rotations, horizontal flips, and random occlusions. These augmentations significantly increase the diversity of the training data, improving the robustness of the deep learning model and reducing the risk of overfitting.

Fig. 3: Augmented face-cropped image: Rotation and brightness adjustment

5) Balancing the Dataset: The augmented dataset is balanced by undersampling the majority class to match the size of the minority class. This step prevents the model from developing a bias toward one category and enhances its ability to generalize effectively. After augmentation and balancing, both real and fake frames are adjusted to include 17,416 frames each, ensuring optimal training conditions and equitable representation of both classes.

C. Training Preparation

The dataset consists of real and fake images, resized to 224x224 pixels for uniform input dimensions. Normalization is applied using ImageNet mean and standard deviation values to standardize pixel intensity. The dataset is split into 64% for training, 20% for validation and 20% for testing, ensuring a balance between model learning and evaluation.

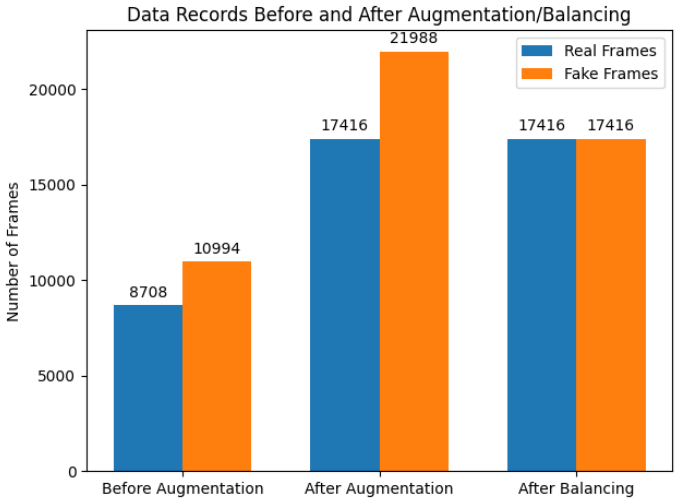


Fig. 4: Numbers of real and fakes frame before augmentation

IV. Shallow Feed Forward

A. Model Architecture

We implemented a Shallow Neural Network (Shallow NN) with hidden dimension is 128, first activation is ReLU, the second is Sigmoid for deepfake detection. Output is 1 dimension since this is a binary classification task.

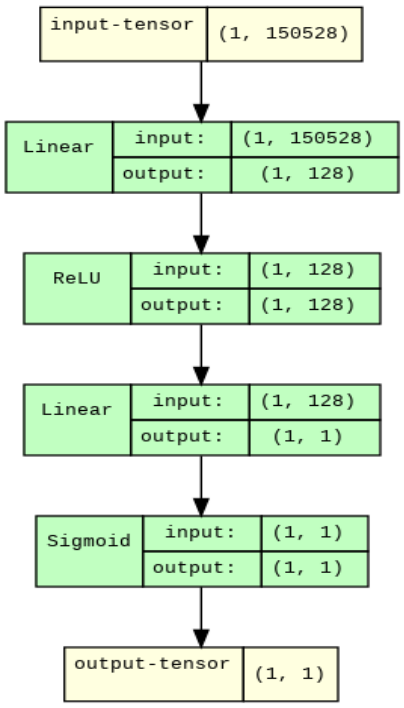
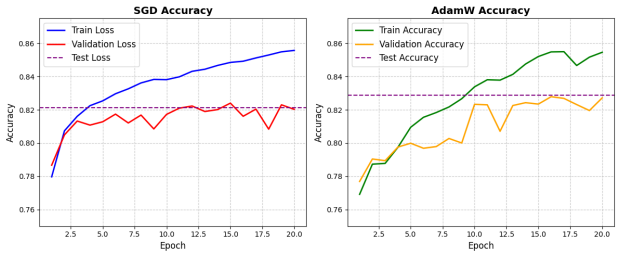


Fig. 5: Shallow Network Architecture

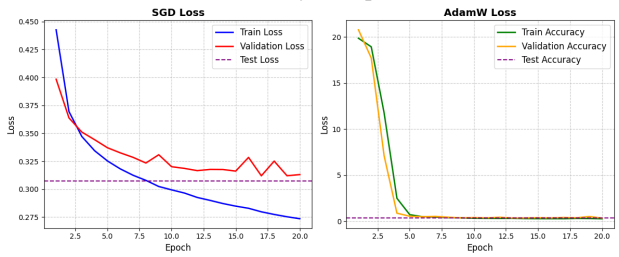
B. Training Results

Initially, a pre-optimized model was trained with SGD optimizer, learning rate = 0.001, and no normalization. This setup achieved a train accuracy of 85.57% and a test accuracy of 82.11% after 20 epochs, with a training time of approximately 25 minutes on a P100 GPU.

To improve performance, we switched to the Adam with weight-day optimizer. This optimization increased train accuracy to 85.45% and resulted in a test accuracy of 82.89%, with a training time of approximately 25 minutes.



(a) Accuracy Comparison



(b) Loss Comparison

Fig. 6: Comparison of SGD and AdamW

C. Stochastic Gradient Descent and Adam Weight-decay

TABLE I: Comparison of Pre-optimized and Optimized Shallow Models

Metric	Pre-optimized (SGD)	Optimized (AdamW)
Optimizer	SGD	AdamW
Weight Decay	No	0.01
Train Accuracy	85.57%	88.29%
Test Accuracy	82.11%	82.89%

After optimization, we observed faster convergence, along with a small improvement in both training accuracy and validation/test accuracy, as well as a reduction in test loss. These results suggest that the adjustments to the model's complexity and learning parameters helped accelerate the training process, while also leading to slight improvements in performance on both the training and unseen data

D. Future Work

To further enhance deepfake detection performance, we plan to experiment with different normalization techniques, explore various loss functions, and refine the network architecture to balance precision and recall more effectively. Additionally, we will investigate techniques like dropout, data augmentation, and early stopping to mitigate overfitting and improve the model's generalization ability.

V. Convolutional Neural Networks

A. Model Architecture

A basic Convolutional Neural Network (CNN) is designed to extract spatial features from images through convolutional layers that apply filters to the input, followed by a ReLU activation for non-linearity. Pooling layers are then applied to reduce the dimensionality of the feature maps. Fully connected layers process the extracted features, and the final classification layer uses a sigmoid activation function to provide a binary classification output. This basic architecture is effective, but further improvements can be made by enhancing

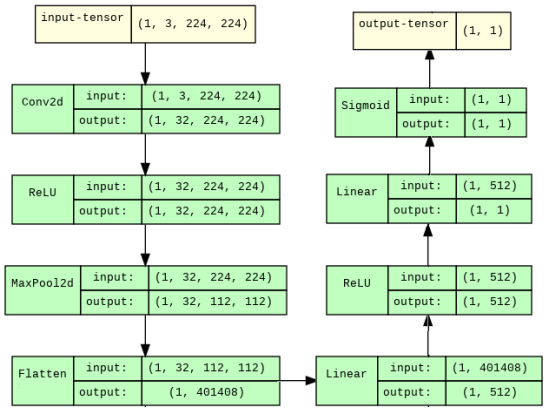
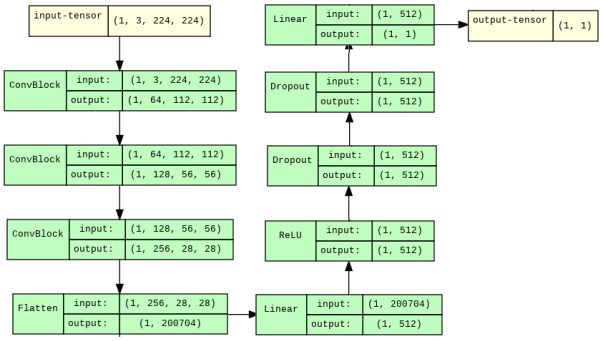
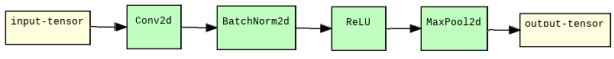


Fig. 7: Traditional CNN Architecture

To enhance the basic model, we added Batch Normalization after each Conv2D layer to stabilize gradients and speed up training. A fully connected layer with 50% dropout prevents overfitting, while max pooling reduces dimensionality. This updated model improves accuracy and generalization.



(a) Updated CNN Architecture



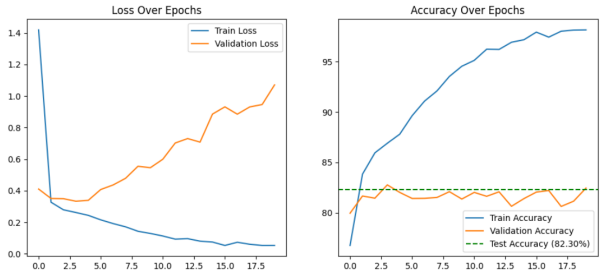
(b) Updated CNN ConvBlock

Fig. 8: Updated CNN Architecture and ConvBlock

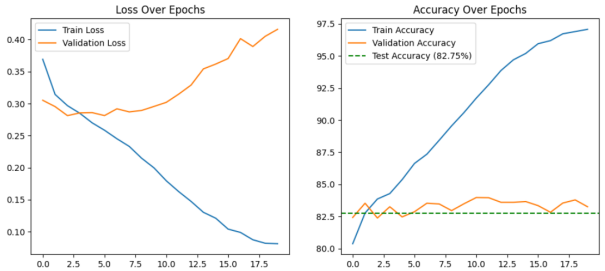
B. Training Results

The traditional CNN, trained with AdamW, achieved high train accuracy (98.15%) but suffered from overfitting, resulting in poor validation (82.44%) and test (82.30%) accuracies. In contrast.

The updated CNN, using a ReduceLROnPlateau scheduler, showed improved generalization. Despite slightly lower train accuracy (97.42%), it achieved better validation (83.48%) and test (83.25%) accuracies, with significantly reduced validation loss, indicating enhanced stability and performance.



(a) Traditional CNN Loss



(b) Updated CNN Loss

Fig. 9: Comparison of CNN Loss: Traditional vs Updated

Metric	Traditional CNN	Updated CNN
Accuracy	98.15%	97.4%
Training Loss	0.0530	0.0741
Validation Accuracy	82.44%	83.48%
Validation Loss	1.0707	0.405
Test Accuracy	82.30%	83.25%

TABLE II: Comparison: Updated CNN Model vs. Traditional CNN

VI. Fine-Tuning CNN Model

To optimize the performance of the CNN model, we conducted a systematic fine-tuning process using the Optuna framework. The tuning procedure involved three phases. The first two phases focus on different hyperparameters, while the third phase re-tunes the most important hyperparameters for the last time.

A total of 10,000 images were used for training and 2,000 images for validation during the tuning process. The result is presented in the table below.

A. Tuning Phases

1) Phase 1: In the first phase, we tuned the **learning rate, optimizer type, batch size and weight decay**, while keeping the numbers of filters, kernel size, dropout rate and activation function constant. The ranges and constant values of the hyperparameters were as followed (hyperparamters with aterisks are those to be kept constant):

Hyperparameter	Values
Learning rate	[1e-5, 1e-2]
Optimizer	SGD, Adam, AdamW
Batch size	16, 32, 64
Weight decay	[1e-5, 1e-2]
Numbers of filters*	(64, 128, 256)
Kernel size*	3
Dropout rate*	0.5
Activation function*	ReLU

TABLE III: Values of hyperparameters for phase 1

The result of the tuning was as in table IV.

2) Phase 2: The second phase optimized **the number of filters in each convolutional layer, the kernel size, dropout rate and activation function**, and used the best hyperparaters values found in Phase 1.

The result is presented in Table VI.

3) Phase 3: The third phase re-tunes the most sensitive hyperparameters: **learning rate, weight decay and dropout rate**, with a smaller search space around the best values found in Phase 1 and Phase 2.

The result is presented in Table VIII.

B. Training and Evaluation

After determining the best hyperparameters, the model was trained for 20 epochs.

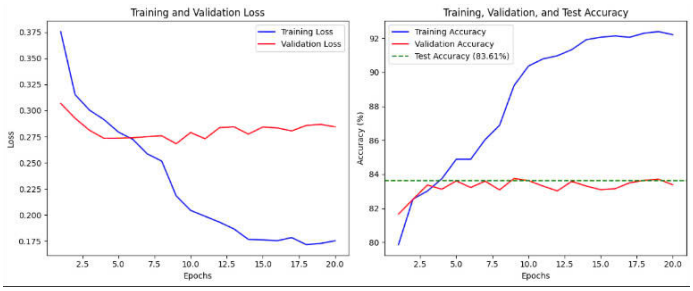


Fig. 10: Losses and Accuracies

The analysis of the loss and accuracy plots reveals several key observations. Firstly, the model exhibits signs of overfitting, as evidenced by the diverging trends between the training and validation metrics. While the training loss consistently decreases and the training accuracy steadily increases, the validation loss plateaus and even increases after approximately 3 epochs, with a corresponding stagnation in validation accuracy. This suggests that the model is beginning to memorize the training data rather than learning generalizable patterns.

Furthermore, a notable generalization gap is present, indicated by the difference between the training accuracy and

Learning rate	Optimizer	Momentum	Batch Size	Weight Decay	Val loss	State
2.46E-03	Adam	nan	16	8.56E-04	0.3324	COMPLETE
2.93E-04	SGD	7.25E-01	16	9.04E-04	0.3465	COMPLETE
1.35E-03	SGD	7.78E-01	64	4.50E-04	0.3451	COMPLETE
1.20E-05	SGD	9.09E-01	32	4.07E-03	0.3116	COMPLETE
5.61E-05	Adam	nan	32	1.58E-04	0.3370	COMPLETE
1.79E-04	AdamW	nan	32	1.55E-04	0.3075	COMPLETE
5.46E-04	AdamW	nan	16	1.19E-04	0.3389	PRUNED
4.88E-05	SGD	9.18E-01	64	2.71E-03	0.3479	PRUNED
3.24E-04	AdamW	nan	32	4.63E-04	0.3220	COMPLETE
1.89E-05	AdamW	nan	32	1.27E-03	0.3371	COMPLETE
5.79E-05	Adam	nan	32	2.08E-05	0.3164	PRUNED
5.03E-03	AdamW	nan	32	1.56E-05	0.3729	PRUNED
1.16E-05	SGD	9.87E-01	32	7.70E-03	0.3172	PRUNED
1.40E-05	SGD	9.83E-01	32	6.34E-05	0.3473	PRUNED
1.43E-04	AdamW	nan	32	7.57E-03	0.3117	COMPLETE
1.64E-04	AdamW	nan	32	9.56E-03	0.3146	COMPLETE
1.17E-04	AdamW	nan	64	4.83E-05	0.3307	COMPLETE
1.22E-03	SGD	8.57E-01	64	2.49E-03	0.3149	PRUNED
2.55E-05	SGD	8.75E-01	32	5.47E-05	0.3611	PRUNED
7.27E-03	AdamW	nan	64	2.07E-04	0.3620	PRUNED
1.01E-04	Adam	nan	16	3.86E-05	0.3397	COMPLETE
6.28E-04	Adam	nan	16	2.86E-03	0.3702	PRUNED
9.54E-05	AdamW	nan	32	4.77E-03	0.3177	PRUNED
2.07E-04	AdamW	nan	32	5.39E-03	0.3374	PRUNED
3.13E-05	AdamW	nan	32	1.21E-03	0.3152	PRUNED
9.14E-05	AdamW	nan	32	4.71E-03	0.3143	COMPLETE
5.14E-04	AdamW	nan	32	1.03E-04	0.3180	PRUNED
3.57E-05	SGD	7.97E-01	32	2.73E-04	0.3741	PRUNED
9.20E-04	AdamW	nan	32	4.42E-03	0.3509	PRUNED
2.78E-03	AdamW	nan	32	1.67E-03	0.3504	PRUNED
2.25E-03	SGD	9.20E-01	32	1.59E-03	0.3191	PRUNED
2.75E-04	SGD	9.14E-01	16	5.58E-04	0.3166	PRUNED
1.42E-04	AdamW	nan	32	6.78E-03	0.3431	PRUNED
7.85E-05	AdamW	nan	32	6.83E-03	0.3102	COMPLETE
7.98E-05	AdamW	nan	32	9.96E-03	0.3170	COMPLETE
2.48E-04	AdamW	nan	32	6.61E-04	0.3139	PRUNED
3.06E-04	Adam	nan	32	6.77E-04	0.3315	PRUNED
1.97E-05	SGD	8.13E-01	16	3.56E-03	0.3766	PRUNED
5.72E-05	SGD	8.09E-01	16	2.92E-03	0.3550	PRUNED
3.77E-04	AdamW	nan	64	3.27E-04	0.3205	PRUNED
4.42E-04	AdamW	nan	64	3.72E-04	0.3357	PRUNED
1.86E-04	Adam	nan	32	1.11E-04	0.3168	PRUNED
1.84E-04	AdamW	nan	32	5.98E-03	0.3121	PRUNED
7.22E-05	AdamW	nan	32	6.37E-03	0.3314	PRUNED
4.59E-05	AdamW	nan	32	2.08E-03	0.3194	PRUNED
4.67E-05	AdamW	nan	32	2.15E-03	0.3479	PRUNED
1.17E-04	AdamW	nan	32	3.49E-03	0.3115	PRUNED
1.15E-04	AdamW	nan	32	3.67E-03	0.3387	COMPLETE

TABLE IV: Phase 1 Hyperparameter Tuning Results
pruned = Automatically terminated because the result is getting worse than the median of the previous trials.
yellow = Best result.

Hyperparameter	Value
First Convolutional Layer Filters	32, 64, 128
Second Convolutional Layer Filters	32, 64, 128
Third Convolutional Layer Filters	32, 64, 128
Kernel Size	3, 5
Dropout Rate	[0.2, 0.7]
Activation	ReLU, LeakyReLU
Learning rate*	1.79e-4
Optimizer*	AdamW
Batch size*	32
Weight decay*	1.55e-4

TABLE V: Values of hyperparameters for Phase 2

the validation/test accuracy. This gap highlights the model's struggle to generalize its learning to unseen data.

Despite this, the model achieves a respectable test accuracy of **83.61%**, higher than the pre-tuned state, suggesting reasonable learning capacity. However, there is room for improvement.

VII. Residual Networks Architecture

A. Model Architecture

ResNet, or Residual Networks, is a deep learning architecture that leverages residual connections to allow for the training of very deep networks. This is done by adding shortcut connections that bypass one or more layers, enabling the

Conv filters	Kernel	Dropout	Activation	Val loss	State
32, 128, 128	5	2.92e-01	LeakyReLU	0.3279	COMPLETE
32, 64, 512	3	2.91e-01	LeakyReLU	0.3310	COMPLETE
128, 128, 128	3	3.24e-01	LeakyReLU	0.3326	COMPLETE
32, 256, 256	5	6.46e-01	LeakyReLU	0.3029	COMPLETE
64, 128, 512	5	6.00e-01	ReLU	0.2984	COMPLETE
64, 64, 512	3	4.67e-01	ReLU	0.3161	COMPLETE
64, 128, 256	5	6.33e-01	LeakyReLU	0.3022	COMPLETE
64, 128, 256	5	3.55e-01	LeakyReLU	0.3452	PRUNED
64, 64, 256	5	4.21e-01	LeakyReLU	0.3616	PRUNED
32, 256, 128	5	5.98e-01	LeakyReLU	0.3141	PRUNED
32, 64, 128	3	3.32e-01	LeakyReLU	0.3320	PRUNED
64, 128, 512	5	5.39e-01	ReLU	0.3142	PRUNED
64, 128, 512	5	5.52e-01	ReLU	0.3433	PRUNED
64, 128, 512	5	6.99e-01	ReLU	0.3355	PRUNED
128, 128, 256	5	6.85e-01	ReLU	0.3497	PRUNED
128, 128, 256	5	4.81e-01	ReLU	0.3373	PRUNED
64, 256, 512	5	5.91e-01	ReLU	0.3368	PRUNED
64, 256, 512	5	5.95e-01	ReLU	0.3340	PRUNED
64, 128, 256	3	4.16e-01	LeakyReLU	0.3194	PRUNED
64, 128, 256	3	2.33e-01	ReLU	0.3142	PRUNED
128, 128, 512	5	5.07e-01	LeakyReLU	0.3314	PRUNED
128, 128, 512	5	5.20e-01	LeakyReLU	0.2992	COMPLETE
32, 256, 256	5	6.47e-01	LeakyReLU	0.3222	PRUNED
32, 256, 256	5	6.48e-01	LeakyReLU	0.3378	PRUNED
32, 256, 256	5	6.29e-01	LeakyReLU	0.3388	PRUNED
32, 256, 256	5	5.74e-01	LeakyReLU	0.3371	PRUNED
32, 256, 256	5	5.28e-01	LeakyReLU	0.3024	COMPLETE
128, 128, 512	5	5.15e-01	ReLU	0.3064	PRUNED
128, 128, 512	5	4.20e-01	LeakyReLU	0.3398	PRUNED
128, 128, 512	5	5.11e-01	ReLU	0.3010	PRUNED
128, 128, 128	5	3.85e-01	ReLU	0.3175	PRUNED
128, 128, 128	5	3.75e-01	LeakyReLU	0.3316	PRUNED
32, 128, 512	5	5.47e-01	LeakyReLU	0.3320	PRUNED
64, 64, 256	5	4.88e-01	LeakyReLU	0.3072	PRUNED
64, 64, 256	5	4.63e-01	LeakyReLU	0.3380	PRUNED
32, 256, 512	3	4.98e-01	LeakyReLU	0.3174	PRUNED

TABLE VI: Phase 2 Hyperparameter Tuning Results

Hyperparameter	Value
Learning Rate	[8.95e-5, 2.69e-4]
Weight Decay	[7.75e-5. 2.33e-4]
Dropout Rate	[3.00e-1, 9.00e-1]
Optimizer*	AdamW
Batch Size*	32
Conv Filters*	(64, 128, 512)
Kernel Size*	5
Activation*	ReLU

TABLE VII: Values of hyperparameters for Phase 3

network to learn residual mappings instead of direct mappings. The ResNet architecture is particularly effective for tasks such as image classification and object recognition.

In our deepfake detection model, we use a variant of ResNet, specifically ResNet-18, which consists of 18 layers. The architecture is designed to learn complex features of both real and fake faces. The ResNet model includes the following key components:

- **Convolutional Layers:** These layers extract low-level features from the input images.
- **Residual Blocks:** These blocks enable the network to learn identity functions, making it easier to train deeper networks. Each block consists of two or three convolutional layers with skip connections.
- **Fully Connected Layer:** After the residual blocks, the network is flattened, and a fully connected layer is used to classify the input as either real or fake.

The primary advantage of using ResNet is its ability to prevent the vanishing gradient problem by allowing gradients to flow directly through the shortcut connections during back-propagation. This results in faster convergence and improved accuracy, which is particularly important when dealing with large and complex datasets like the deepfake dataset.

B. Training Results

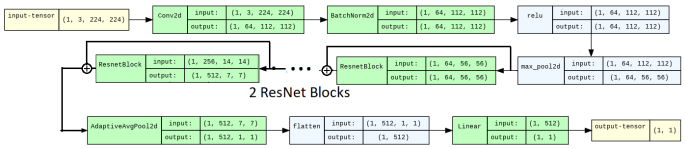
The ResNet model achieved a training loss of 0.1964 and a training accuracy of 88.92%. This demonstrates the model's ability to learn the patterns in the training data effectively.

For the validation, the model showed a validation loss of 0.3232 and a validation accuracy of 82.93%. The validation accuracy is slightly lower than the training accuracy, which is typical for most machine learning models and suggests that the model is generalizing well to unseen data without significant overfitting.

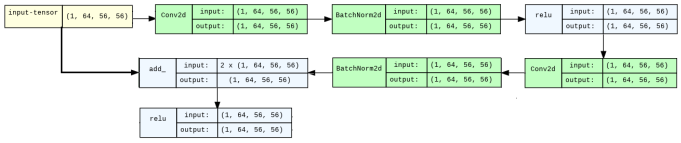
The model achieved a test accuracy of 83.50%. The test accuracy is close to the validation accuracy, suggesting that the

lr	Weight decay	Dropout	Val loss	State
2.30E-04	1.44E-04	5.94E-01	0.3133	COMPLETE
1.71E-04	1.27E-04	4.38E-01	0.2996	COMPLETE
2.04E-04	8.18E-05	4.33E-01	0.3039	COMPLETE
2.05E-04	1.17E-04	5.07E-01	0.3047	COMPLETE
2.55E-04	1.60E-04	4.38E-01	0.3024	COMPLETE
1.57E-04	7.94E-05	5.80E-01	0.3053	COMPLETE
1.30E-04	2.14E-04	5.48E-01	0.3054	COMPLETE

TABLE VIII: Phase 3 Hyperparameter Tuning Results



(a) ResNet Overview Architecture



(b) ResNet Block Architecture

Fig. 11: ResNet Architecture

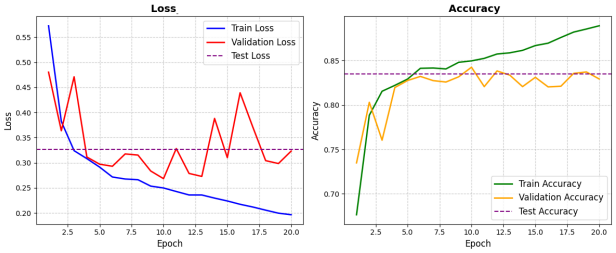


Fig. 12: ResNet Training Results

model is not overfitting to the training data and is generalizing well to new, unseen data.

ResNet Classification Report

	Predicted: 0	Predicted: 1
Actual: 0	TP = 2992	FP = 492
Actual: 1	FN = 658	TN = 2826

Class	Precision	Recall	F1-Score	Support
0.0	0.82	0.86	0.84	3484
1.0	0.85	0.81	0.83	3484
Accuracy			0.83	6968
Macro avg	0.84	0.83	0.83	6968
Weighted avg	0.84	0.83	0.83	6968

VIII. Efficient Net B0

A. Model Architecture

EfficientNetB0 is a highly efficient CNN designed for image classification. It consists of the following key components:

Stem Layer

- Conv2d Layer: Initial convolution with 3x3 kernel and 32 channels.
- Batch Normalization and Swish Activation.

MBConv Blocks

- Expansion, Depthwise, and Pointwise Convolutions.
- Residual Connections whenever possible.
- Batch Normalization and Swish Activation.

Head and Pooling Layer

- 1x1 Convolution, Batch Normalization, Swish Activation.
- Adaptive Average Pooling.

Classifier

- Linear Layer: Reduces features to 1 output.
- Sigmoid Activation.

The Swish activation function, defined as:

$$\text{Swish}(x) = x \cdot \frac{1}{1 + e^{-x}}$$

is similar to ReLU but employs a smooth, non-monotonic curve instead of a hard threshold at zero. This function also benefits from a simple derivative.

$$\text{Swish}'(x) = \sigma(x) + x\sigma(x)(1 - \sigma(x))$$

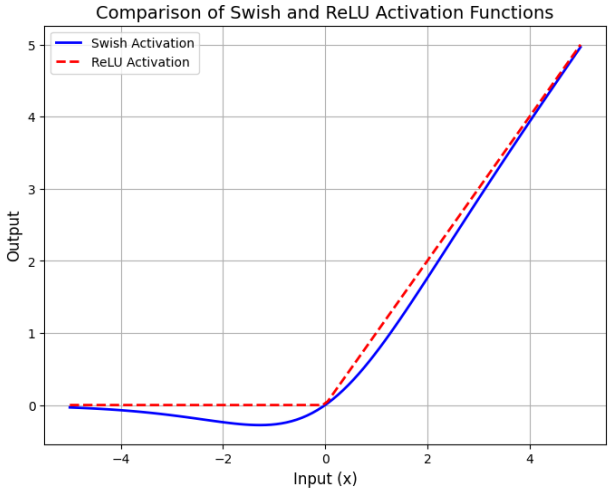
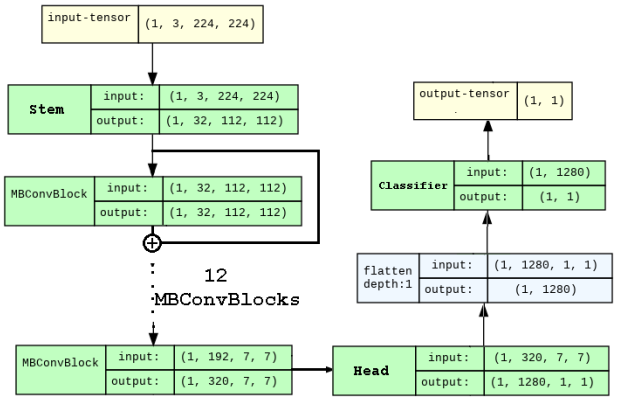
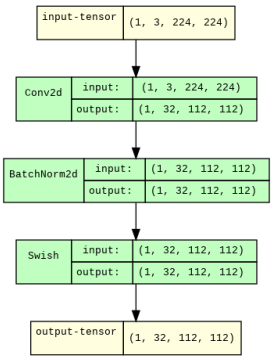


Fig. 13: Comparison of Swish and ReLU Activation Functions

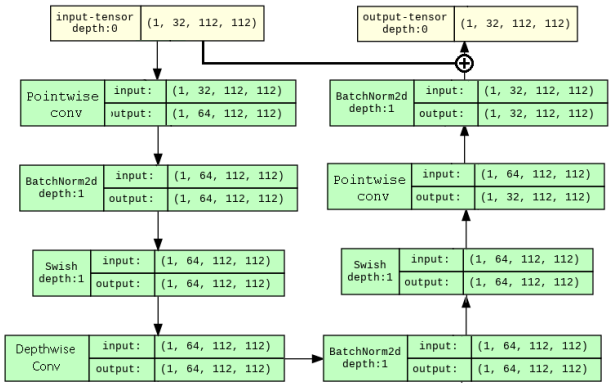
EfficientNetB0 allows us to balance between performance and efficiency, achieving high accuracy with fewer parameters.



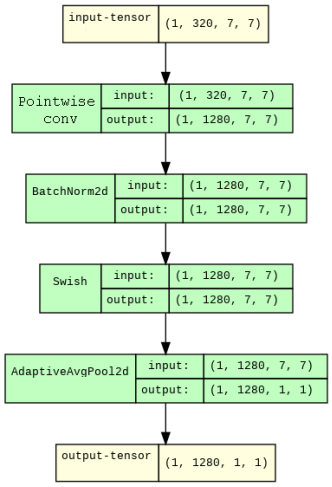
(a) EfficientNet Overview



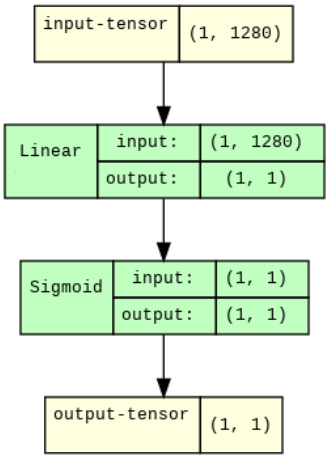
(b) EfficientNet Stem Architecture



(c) EfficientNet MBConv Architecture



(d) EfficientNet Head Architecture



(e) EfficientNet Classifier Architecture

Fig. 14: EfficientNet Architecture

B. Training Results

The EfficientNet B0 model's training exhibited a consistent decrease in loss and a corresponding increase in accuracy, signaling effective learning on the training data. However, the validation phase revealed a more volatile pattern, with fluctuating losses and a plateaued accuracy around 81-82%, suggesting potential overfitting and limited generalization.

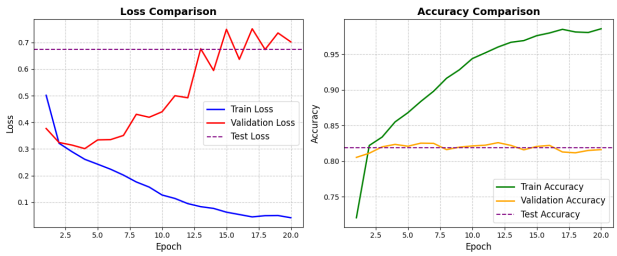


Fig. 15: EfficientNet B0 Loss

EfficientNet Classification Report

	Predicted: 0	Predicted: 1
Actual: 0	TP = 2860	FP = 624
Actual: 1	FN = 642	TN = 2842

Class	Precision	Recall	F1-Score	Support
0.0	0.82	0.82	0.82	3484
1.0	0.82	0.82	0.82	3484
Accuracy			0.82	6968
Macro avg	0.82	0.82	0.82	6968
Weighted avg	0.82	0.82	0.82	6968

IX. Our Custom Model C5

A. Model Architecture

We hypothesize that each convolutional neural network (CNN) will learn unique patterns and produce embedding vectors from images. For our experiment, we employed five different CNNs with varying kernel sizes, strides, and padding. The outputs of these networks were flattened and passed through a self-attention mechanism without positional encoding to combine the features. Finally, two linear layers were used as classifiers along with ReLU activations.

To ensure efficient performance, we limited each CNN to a maximum of three convolutional layers, running them in parallel. Out of the five CNNs, two utilized conventional grain kernels with a ConvNet-like architecture, while the remaining three employed depth-wise separable convolution layers with larger kernel sizes to capture broader patterns. We incorporated the Swish activation function to minimize information loss during processing. Additionally, we experimented with the number of attention layers, implementing models with either one or two encoding layers.

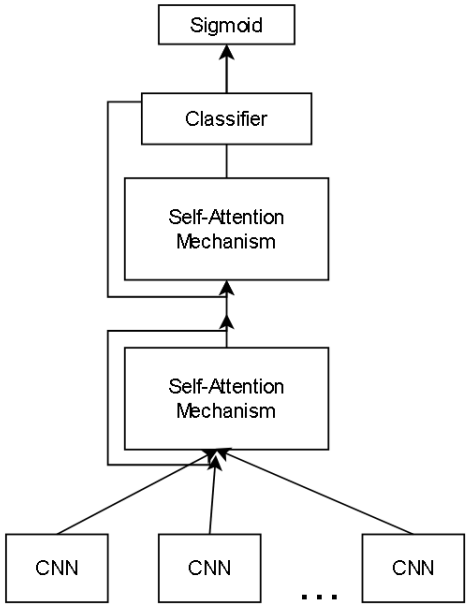
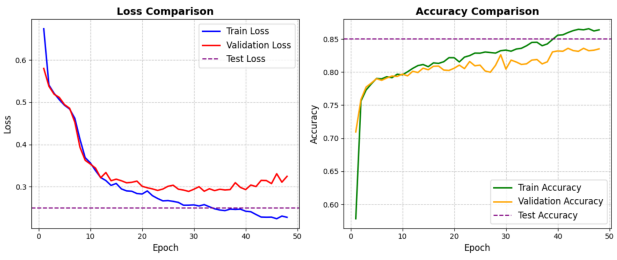


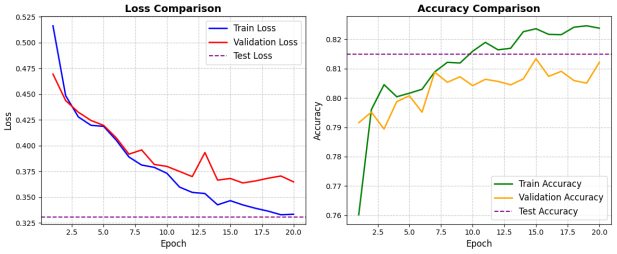
Fig. 16: C5 Architecture

B. Training Results

While training accuracy lower, and validation accuracy are comparable to then EfficientNet B0 and Residual Network and the gap between training and validation accuracies (86.38% and 83.51%) suggests strong generalization. On the test set, the model achieved a loss of 0.36 and an accuracy of 83.55%. C5 with two attention layers has lower accuracy in both validation (81.21%) and testing (0.33 and 81.49%).



(a) C5 Loss with 1 Attention Layer



(b) C5 Loss with 2 Attention Layers

Fig. 17: Comparison of C5 Loss between 1 and 2 Attention Layers

C5 1 Attention Layer Report

	Predicted: 0	Predicted: 1
Actual: 0	TP = 2391	FP = 1093
Actual: 1	FN = 191	TN = 3293

Class	Precision	Recall	F1-Score	Support
0.0	0.93	0.69	0.79	3484
1.0	0.75	0.95	0.84	3484
Accuracy			0.82	6968
Macro av	0.84	0.82	0.81	6968
Weighted avg	0.84	0.82	0.81	6968

C5 2 Attenion Layers Report

	Predicted: 0	Predicted: 1
Actual: 0	TP = 2805	FP = 679
Actual: 1	FN = 611	TN = 2873

Class	Precision	Recall	F1-Score	Support
0.0	0.82	0.81	0.81	3484
1.0	0.81	0.82	0.82	3484
Accuracy			0.81	6968
Macro avg	0.81	0.81	0.81	6968
Weighted avg	0.81	0.81	0.81	6968

C. Discussion

Here, we gives some ideas about why our model results are lower than EfficientNet B0 and ResNet:

CNNs Depth Limitation: Since we focus on saving memory and computational costs, we both limit our model dimensions and CNNs depth, with only 3 convoluational layers at most, the numbers extracted of pattern and information is limited.

Limitation is the choice of CNNs: To make the outputs of CNNs fit with model dimensions, we had to apply big stride and max pooling. Furthermmore, many advanced CNNs like ResNet and EficientNet B0, although our input is inspired by those, we can only do so much with only 3 convolutional layers.

Not enough features for two self-attention layers: We hypothesize that only 5 inputs do not have enough ”deep” information for further extraction, thus the second attention layer is redundant, and cost more memory and computational costs. Therefore, we decided to stop training the two attention layers model and focus on the other one.

X. Model Deployment Real-Time

We developed a real-time deepfake detection website that allows users to identify manipulated videos efficiently. The system uses the state-of-the-art EfficientNet model for deepfake detection and provides users with the option to upload or stream videos from various sources for real-time analysis.

A. Main Page

The website’s main page serves as the hub for users to interact with the deepfake detection system. It contains the following key features:

1) Our Team: On the main page, we introduce our team of developers and researchers who worked on the deepfake detection system. The section provides background information about the members, their roles, and their contributions to the project.

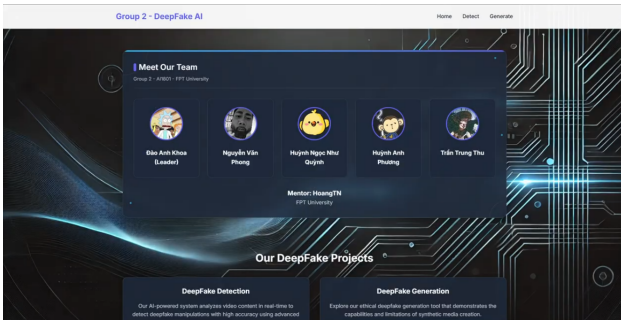


Fig. 18: Our Team Members

2) Site Introduction: The website introduction section explains the purpose and objectives of the platform, highlighting how it can be used to detect deepfakes and ensure the authenticity of digital content. It offers a brief overview of how the deepfake detection process works and the role of the EfficientNet model in providing accurate results.

3) Navigation: Discriminate and Generate: The main page also contains navigation options for users to either discriminate deepfakes or generate results. Users can choose between these two options based on their needs: **Discriminate** and **Generate**

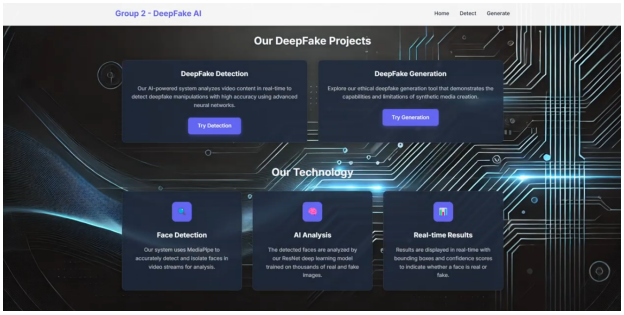


Fig. 19: Navigators

B. Deepfake Discrimination

Users can choose from different input methods to perform deepfake detection

- **YouTube**: Users can input a YouTube video URL, and the system will stream and process the video for deepfake detection.
- **Local Disk**: Users can upload video files in various formats, including MP4, AVI, and MOV, to perform deepfake detection locally.
- **Webcam**: This feature enables users to use their webcam to perform real-time deepfake detection on live footage.

ResNet Model for Deepfake Detection The Resnet model, as trained above, achieved a test accuracy of 83.50% and a validation accuracy of 82.93%. This demonstrates its capability to effectively classify video frames, enabling the website to detect deepfakes in real-time.

Once a video is uploaded or selected, the system processes each frame in real-time using the EfficientNet model to analyze and classify the video as either real or fake.

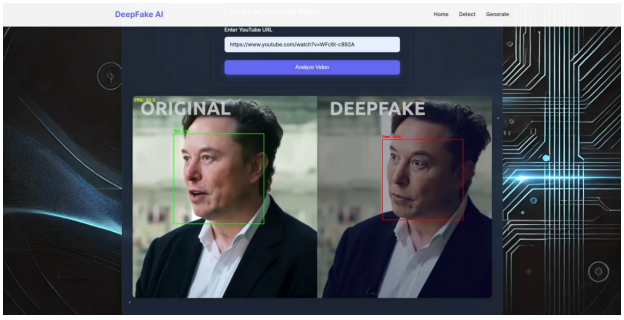


Fig. 20: Prediction Results from the Web Interface

C. Deepfake Generation

Deepfake Generation Using InsightFace For deepfake generation, we utilize the `InsightFace` project, an advanced open-source toolbox for face analysis. It enables real-time face-swapping using state-of-the-art models like `inswapper-512-live`. InsightFace offers robust face detection, alignment, and recognition tools, all optimized for high-quality deepfake creation.

- **Source Face Image**: Users must upload an image of the source face to be used in the deepfake generation process.
- **Target Face Image/Video**: Users can upload a target face image or video onto which the source face will be placed.

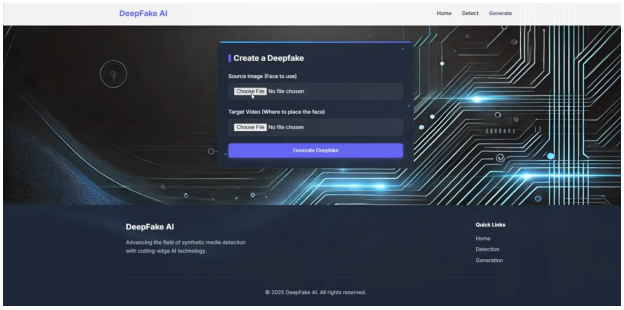


Fig. 21: Deepfake Generation Source Selections

XI. Conclusion

This project demonstrates how deep learning, particularly the ResNet model, can be used to tackle the complex problem of deepfake detection. Through data preprocessing, augmentation, model training, and optimization, we have developed a system capable of identifying manipulated content in real-time.

The performance comparison highlights the effectiveness of different models in terms of test accuracy. Among the models tested, ResNet achieved the highest accuracy, closely

followed by the Updated CNN. Shallow Optimized (AdamW) and Traditional CNN models performed reasonably well, demonstrating consistent reliability. Meanwhile, the Shallow Pre-optimized (SGD) model and our custom model C5 with EfficientNet have showed slightly lower performance. Lastly, the Fine Tuning CNN displayed the lowest accuracy, suggesting room for optimization or further evaluation. This analysis underscores the superior reliability of advanced architectures like ResNet in achieving high

TABLE IX: Comparison of Model Performance

Model	**Test Accuracy**	**F1-Score**	**PyTorch Save Weight**
Fine Tuning CNN	83.60%	0.80	NaN
C5 1 Attention	83.55%	0.815	25.44 MB
ResNet	83.50%	0.830	44.79 MB
Updated CNN	83.25%	NaN	NaN
Shallow Optimized (AdamW)	82.89%	NaN	308.3 MB
Traditional CNN	82.30%	NaN	NaN
Shallow Pre-optimized (SGD)	82.11%	NaN	77.07 MB
EfficientNet B0	82.00%	0.820	12.85 MB
C5 2 Attentions	81.50%	0.815	48.30 MB

The deployment of the model on a website with Flask provides a practical application for users to detect deepfakes. Future work could involve expanding the dataset, improving the model with more sophisticated architectures, and refining the deployment for enhanced performance in real-world scenarios.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2012.

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

[3] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *Proceedings of the 36th International Conference on Machine Learning*, pp. 6105–6114, 2019.