Using Deep Learning to Detect Altered Faces

A Project report

submitted by

RAJA MEHDI ALI KHAN (422244)

YOGESH KUMAR (422275)

NEERUKONDA LAKSHYA VARDHAN(422228)

Under the guidance of **Mr. MANI PRASAD**

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY, ANDHRA PRADESH

TADEPALLIGUDEM - 534101

DECLARATION

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Neerukonda Lakshya vardhan	**Yogesh Kumar**	**Raja Mehdi Ali Khan**
Roll No: 422228	Roll No: 422275	Roll No: 422244
Date: _____	Date: _____	Date: _____

CERTIFICATE

This is to certify that the work contained in the thesis titled **"Using Deep Learning to Detect Altered Faces"**, carried out by **Raja Mehdi Ali Khan** (Roll No: 422244), **Yogesh Kumar** (Roll No: 422275), and **Neerukonda Lakshya vardhan** (Roll No: 422228), has been completed under my supervision.

I further certify that this work has not been submitted elsewhere for the award of any degree or diploma.

Signature

Mr. Mani Prasad

CSED, NIT Andhra Pradesh

April 2025

ACKNOWLEDGEMENTS

We extend our heartfelt gratitude to our supervisor, Mr. Mani Prasad, for his invaluable guidance and support throughout this project. His mentorship has illuminated the path forward, offering clarity and direction as we navigated through challenges and uncertainties. With his steadfast encouragement and wisdom, we were able to overcome obstacles and complete the project successfully. Mr. Mani Prasad's dedication to our growth and development has been truly inspiring, and we are immensely grateful for his unwavering belief in our abilities.

ABSTRACT

In an age where synthetic media threatens the credibility of visual information, we present a robust deepfake detection system that mimics human intuition to unmask digital deception. Our approach fuses the power of **Convolutional Neural Networks (CNNs)** with the temporal sensitivity of **Long Short-Term Memory (LSTM)** networks, enhanced by an **attention mechanism** that spotlights suspicious frames. By extracting rich video features—optical flow, facial landmarks, and deep embeddings—and training on a balanced dataset of real and fake videos, our model learns to detect even the subtlest of forgeries.

We amplify the model's generalization through smart augmentations like flips and jittering, and optimize training using **weighted cross-entropy loss** and **cosine-annealed learning rates**. With a striking **accuracy of 92.86%**, **perfect precision**, and an **AUC-ROC of 1.0**, our system not only excels quantitatively but also provides visual interpretability through attention heatmaps. Misclassifications reveal room for improvement—highlighting the evolving sophistication of deepfakes and the need for multimodal cues. Built and tested on Kaggle with real-world performance in mind, our solution offers a promising step toward safeguarding digital truth.

Contents

1 Introduction

In today's hyper-digital world, faces are more than just identity—they are currency in communication, security, and storytelling. With the rise of social media, photo-editing tools, and AI-powered generators, it has become increasingly easy to alter facial images. Whether it's a simple beauty filter or a highly convincing deepfake, altered faces are now a common part of online content, raising questions about what we can really trust.

The growing sophistication of these manipulations makes them difficult to detect with the naked eye or basic software tools. Traditional methods often fall short when faced with high-resolution, AI-generated fakes that mimic real expressions, lighting, and even micro-expressions. This is where deep learning steps in—powerful neural networks can analyze millions of image features and uncover hidden clues that reveal whether a face has been digitally altered.

This report delves into how deep learning is revolutionizing the way we detect altered faces, from identifying deepfakes in videos to spotting minor edits in photographs. We'll explore the key models and techniques being used, such as CNNs and autoencoders, along with the datasets that train them. In a world where digital deception is evolving rapidly, these technologies play a critical role in protecting truth, privacy, and public trust.

1.1 Problem Statement

With the rapid advancement of generative AI, detecting altered or manipulated facial images has become a critical challenge in digital forensics, cybersecurity, and media integrity. Traditional detection systems struggle to keep up with the complexity and realism of modern deepfakes and AI-edited images, which often evade both human observation and rule-based algorithms. This growing threat undermines trust in digital content, enabling misinformation, identity theft, and even fraud—making it imperative to develop intelligent, scalable, and adaptive solutions that can reliably distinguish authentic faces from manipulated ones.

1.2 Objectives

1. To understand and analyze various types of facial image alterations, including deepfakes, morphing, and subtle photo manipulations.

2. To study and evaluate deep learning techniques (e.g., CNNs, autoencoders, and transformers) for their effectiveness in detecting altered facial content.

3. To develop a robust deep learning model capable of identifying forged or tampered facial images with high accuracy across diverse datasets.

4. To test the model's performance on both real and synthetic images and assess its generalization to new types of manipulations.

5. To propose a scalable and practical solution that can be integrated into digital platforms for real-time or large-scale altered face detection.

1.3 Scope of the Project

This project focuses on leveraging deep learning techniques to detect altered or manipulated facial images with a high degree of accuracy and reliability. The scope includes identifying various forms of face tampering, such as deepfakes, facial reenactment, morphing, and other AI-based modifications. The system will be trained and evaluated using publicly available datasets containing both real and altered facial images. The project also encompasses the implementation of a suitable deep learning architecture—such as Convolutional Neural Networks (CNNs) or transformer-based models—and the development of a pipeline for pre-processing, feature extraction, classification, and result interpretation. While the primary focus is on image-based detection, the framework may be extended in the future to video analysis and real-time applications. The goal is to contribute toward building a reliable tool that can assist in digital forensics, content moderation, and identity verification systems.

2 Literature Review

2.1 Deepfake Challenges

In today's digital world, where videos are widely shared on social media and news platforms, deepfakes—videos with AI-altered faces—pose a growing threat to trust, security, and privacy. The following points highlight the challenges:

- Rising Threat of Deepfakes: Advanced AI tools make deepfakes increasingly realistic and easy to create, allowing anyone to produce convincing fake videos. This fuels risks in misinformation, political manipulation, and identity theft.

- Challenges in Manual Detection: Human viewers struggle to spot deepfakes, as they mimic natural facial movements, lighting, and speech. Traditional forensic methods, like checking pixel errors, often fail against sophisticated AI-generated videos.

- Social and Legal Ramifications: Without effective automated detection, deepfakes can spread rapidly, causing reputational harm, fake news, and cybercrimes. Legal systems lack tools to address these issues swiftly, underscoring the need for technological solutions.

2.2 Existing Deepfake Detection Methods and Limitations

Several approaches have been developed to detect deepfakes in videos, but they face significant limitations:

1. Manual forensic analysis spots lighting or pixel noise inconsistencies but fails when deepfakes use advanced rendering or videos are compressed.

2. Classical machine learning (e.g., SVMs or decision trees) relies on hand-crafted features like blink patterns, which are ineffective against modern, diverse deepfake methods.

3. CNN-based models analyze frame details but struggle with temporal inconsistencies in videos and require large datasets for training.

4. 3D CNNs and transformers capture spatial and temporal patterns but demand high computational resources, making them impractical for beginners or resource-constrained platforms like Kaggle.

5. Dataset-specific solutions trained on limited datasets fail to generalize to new manipulation techniques or unseen video types.

2.3 Bridging the Gap

Deep learning offers a promising solution to overcome the shortcomings of existing deepfake detection methods, particularly through hybrid architectures like CNN-LSTM models. Our project leverages a CNN-LSTM model with an attention mechanism to detect altered faces in videos, combining the strengths of CNNs for analyzing frame details (e.g., facial textures) and LSTMs for tracking temporal changes (e.g., unnatural movements). Features like data augmentation enhance generalization with a small dataset of approximately 140 videos, while attention visualization provides transparency into model decisions. Implemented in Kaggle, this approach is beginner-friendly, requiring minimal manual feature engineering and adapting to various deepfake types. By addressing generalization, computational efficiency, and interpretability, our system paves the way for reliable, automated deepfake detection, crucial for media platforms, security, and public trust.

3 Methodology

To build our deepfake detection system, we designed a pipeline that processes videos, extracts clues, enhances data, builds a smart model, and trains it to spot fakes. Here's how we did it, step by step, using tools like Kaggle, PyTorch, and OpenCV to make it work.

3.1 Data Acquisition

Getting the right videos and preparing them was our first step. We needed a dataset that had both real and fake videos to teach our model the difference.

3.1.1 Video Dataset

We used a dataset from Kaggle with around 140 videos, split evenly between 70 real and 70 fake ones, stored in `/kaggle/input/videos/videos/real` and `/fake`. Real videos show genuine human faces, while fake ones have AI-altered faces, like swapped identities or changed expressions. Each video is a short clip, perfect for testing our model. We loaded these using OpenCV's `VideoCapture`, making sure our code could handle different video formats and lengths.

3.1.2 Labeling and Preprocessing

To train the model, we labeled real videos as 0 and fake ones as 1, creating a clear binary classification task. Preprocessing involved resizing frames to 224x224 pixels to standardize input and sampling up to 30 frames per video, skipping every 5th frame to save computation. This was done using our `VideoDataset` class, which organizes videos and labels for easy loading. We also split the data into 80% training (112 videos) and 20% testing (28 videos) using `train_test_split` to ensure balanced classes.

3.2 Feature Extraction

To spot deepfakes, we needed to pull out clues from each video frame that might reveal fakes, like odd movements or face shapes. We extracted four types of features, which we combined into a single vector for each frame.

3.2.1 Optical Flow Analysis

Optical flow tracks how pixels move between frames, like catching unnatural lip twitches. We used OpenCV's Farneback algorithm to compute motion vectors. For two consecutive frames I_t and I_{t+1}, the optical flow $\mathbf{u} = (u_x, u_y)$ is estimated by solving:

$$I_t(x, y) = I_{t+1}(x + u_x, y + u_y). \tag{1}$$

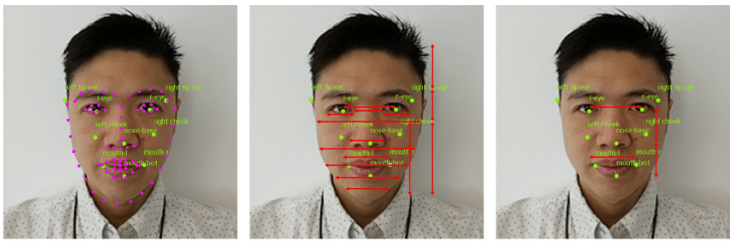
This gives a 2D vector per pixel, which we averaged over the frame to get a compact feature, stored via `compute_optical_flow`. It's great for spotting jerky or fake motion in deepfakes.

3.2.2 Facial Landmarks Detection

Facial landmarks mark key face points, like eyes and nose, to check for weird shapes. Using Dlib's 68-point predictor, we detected coordinates (x_i, y_i) for $i = 1, \ldots, 68$, forming a vector:

$$\mathbf{L} = [(x_1, y_1), (x_2, y_2), \ldots, (x_{68}, y_{68})]. \tag{2}$$

Our `extract_facial_landmarks` function outputs a 136D vector (68 x-coordinates, 68 y-coordinates) per frame. This helps catch distortions, like misaligned jaws in fake videos.



(a) (b) (c)

Figure 1: Example of 68 facial landmarks detected on a video frame, showing key points like eyes and mouth.

3.2.3 Deep Feature Extraction

To capture complex patterns, like fake textures, we used a pretrained EfficientNet-B0 model from the TIMM library. For each frame, we extracted a feature vector $\mathbf{F}$ from the last convolutional layer, typically 1280D, using `extract_deep_features`. This is like borrowing a pro's eyesight to spot subtle clues humans miss, making our model more powerful without needing tons of data.

3.2.4 Frame Difference Computation

Frame differences highlight changes between frames, like sudden color shifts in fakes. For frames I_t and I_{t+1}, we computed the absolute difference:

$$D(x, y) = |I_{t+1}(x, y) - I_t(x, y)|, \tag{3}$$

averaging pixel differences across the frame using `cv2.absdiff`. This gives a compact feature that catches inconsistencies, like flickering in deepfake faces. All features are combined into a single vector per frame, padded to a fixed length, and averaged across frames using `extract_video_features`.

3.3 Data Augmentation

With only 140 videos, we needed to make our model robust to different conditions, like lighting or angles, without collecting more data.

3.3.1 Frame Transformation Techniques

We applied random transformations to frames using our `VideoAugmenter` class, including horizontal flips, small rotations (up to 10 degrees), color jitter (brightness, contrast), and random erasing of frame patches.

This simulates real-world variations, helping the model generalize. For a frame I, a flip transforms pixel (x, y) to $(W - x, y)$, where W is the width, and jitter adjusts pixel values randomly. These tricks make the model think it's seeing more videos, boosting its ability to spot fakes in new settings.

3.4 Model Architecture

The brain of our system is the `CNNLSTMDetector`, a model that combines Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTM) networks, and an attention mechanism to analyze videos.

3.4.1 CNN-LSTM Framework

The CNN part processes each frame's feature vector, looking for spatial clues like odd textures. It has three 1D convolutional layers with 96, 192, and 384 filters, each followed by batch normalization, ReLU activation, and max-pooling. The LSTM part, with 2 bidirectional layers and 192 hidden units, tracks how these clues change over time, like catching fake blinks. For a sequence of frame features $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T]$, the LSTM outputs hidden states $\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_T]$. This setup is perfect for videos, as it sees both still images and motion.

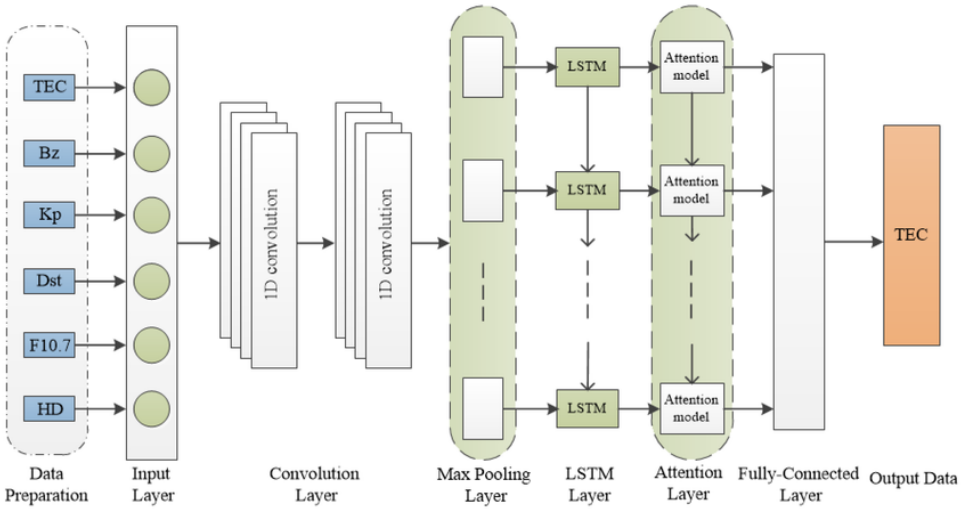


Figure 2: Architecture of the CNN-LSTM model, showing feature extraction, CNN layers, LSTM units, and attention mechanism.

3.4.2 Attention Mechanism

To focus on key frames, like ones with weird movements, we added an attention mechanism. It assigns weights α_t to each frame's hidden state $\mathbf{h}_t$, computed as:

$$e_t = \mathbf{w}^T \tanh(\mathbf{W}\mathbf{h}_t + \mathbf{b}), \quad \alpha_t = \frac{\exp(e_t)}{\sum_{j=1}^{T} \exp(e_j)}, \tag{4}$$

where $\mathbf{W}$, $\mathbf{w}$, and $\mathbf{b}$ are learned parameters. The weighted sum $\mathbf{c} = \sum_{t=1}^{T} \alpha_t \mathbf{h}_t$ feeds into fully connected layers with dropout (0.5) to predict real or fake. This makes the model smarter by prioritizing important moments, and the weights help us see what it's looking at.

3.5 Training Process

Training the model was like teaching it to spot fakes by showing it examples and tweaking its guesses until it got better.

3.5.1 Loss Function and Optimization

We used a weighted cross-entropy loss to penalize mistakes, especially on real videos, since missing a fake is worse than flagging a real one. For predictions $\hat{y}$ and true labels $y \in \{0, 1\}$, the loss is:

$$L = -\frac{1}{N} \sum_{i=1}^{N} \left[w_0 y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right], \tag{5}$$

where $w_0 = 1.3$ weights real videos, and N is the batch size (16). We optimized this using AdamW with a learning rate of 0.0001 and weight decay of 0.0001, implemented in `train_epoch`. This helped the model learn fast and avoid overfitting.

3.5.2 Learning Rate Scheduling

To make training smoother, we used Cosine Annealing to adjust the learning rate over time. The learning rate η_t at epoch t is:

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min}) \left(1 + \cos\left(\frac{t}{T_{max}} \pi \right) \right), \tag{6}$$

where $\eta_{max} = 0.0001$, $\eta_{min} = 0.00001$, and $T_{max} = 20$ epochs. This starts high, then lowers gently, helping the model converge. We trained for up to 20 epochs, stopping early if validation loss didn't improve for 3 epochs, saving the best model to `/kaggle/working/best_model.pth`.

4 Experimental Setup and Results

We set up our experiments to test how well our deepfake detection system could spot fake videos, using the tools and methods we built. Below, we explain how we prepared our data, ran our pipeline, and checked the results, sharing what we learned along the way.

4.1 Dataset Preparation

To get our model ready, we processed our video dataset and organized it for training and testing. This step was crucial to make sure our model could learn from the right clues.

4.1.1 Video Feature Extraction

We started with 140 videos from Kaggle, with 70 real and 70 fake, as shown in our output: *Found 70 real videos and 70 fake videos for training.* Using our `extract_video_features` function, we processed each video to extract features like optical flow, facial landmarks, deep features, and frame differences, as described in Section 3.2. Processing took about 2 minutes 29 seconds for real videos and 1 minute 55 seconds for fake videos, due to varying video lengths. After combining features, we standardized them to create a dataset with shape $(140, 30, 252296)$, where each video has 30 frames, and each frame has a 252296-dimensional feature vector. The labels formed a vector of shape $(140,)$, with a balanced class distribution: 70 real (0) and 70 fake (1).

4.1.2 Train-Test Split

We split the dataset into 80% training (112 videos) and 20% testing (28 videos) using `train_test_split`, ensuring an even mix of real and fake videos in both sets. Our output confirmed: *Training set size: 112, Test set size: 28.* This split let us train our model on most of the data while keeping a separate set to check how well it generalizes to new videos.

4.2 Workflow

Our workflow tied together all the pieces of our pipeline, from loading videos to making predictions. We began by loading videos with our `VideoDataset` class, which fed them into `DataLoader` for batch processing. Next, we extracted features using `extract_video_features`, applied augmentations via `VideoAugmenter`, and trained our `CNNLSTMDetector` model over multiple epochs. After training, we evaluated performance on the test set with `validate_epoch` and made single-video predictions using `predict_video`. We also visualized attention weights with `visualize_attention` to understand our model's decisions. This end-to-end process ensured we could test our system thoroughly.

4.3 Implementation Details

We built and ran everything in Kaggle's cloud environment, which gave us free access to GPUs for faster training. We used Python 3.8 with key libraries:

- **PyTorch**: For building and training our CNN-LSTM model.
- **OpenCV**: For video loading and optical flow computation.
- **Dlib**: For facial landmark detection.
- **TIMM**: For pretrained EfficientNet-B0 features.
- **Scikit-learn**: For metrics and train-test splitting.

Our code, organized in a Kaggle notebook, included classes like `VideoDataset`, `VideoAugmenter`, and `CNNLSTMDetector`, with functions for feature extraction (`extract_facial_landmarks`, `compute_optical_flow`), training (`train_epoch`), and evaluation (`debug_test_predictions`). We saved outputs like models (`best_model.pth`) and plots (`training_metrics.png`) to `/kaggle/working`.

4.4 Evaluation Metrics

To measure how well our model performed, we used both numbers and visual insights to get a full picture of its strengths and weaknesses.

4.4.1 Quantitative Metrics

We calculated five metrics to judge our model's performance:

- **Accuracy**: The fraction of correct predictions:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \tag{7}$$

  where TP = true positives, TN = true negatives, FP = false positives, FN = false negatives.

- **Precision**: The fraction of predicted fakes that were actually fake:

$$Precision = \frac{TP}{TP + FP}. \tag{8}$$

- **Recall**: The fraction of actual fakes correctly identified:

$$Recall = \frac{TP}{TP + FN}. \tag{9}$$

- **F1 Score**: The harmonic mean of precision and recall:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}. \tag{10}$$

- **AUC-ROC**: The area under the receiver operating characteristic curve, measuring how well the model separates real and fake videos.

These metrics helped us quantify our model's accuracy and reliability.

4.4.2 Qualitative Analysis

For a deeper understanding, we looked at visual outputs. We plotted attention weights with `visualize_attention` to see which frames the model focused on, helping us check if it was catching the right clues. We also analyzed misclassified videos using `debug_test_predictions` to understand why our model made mistakes, looking at frame quality or feature patterns. These visuals made our model's decisions more transparent, especially for beginners like us.

4.5 Results

We were excited to see how our model performed after all our hard work. Here's what we found, using the outputs from our Kaggle notebook.

4.5.1 Model Performance on Test Set

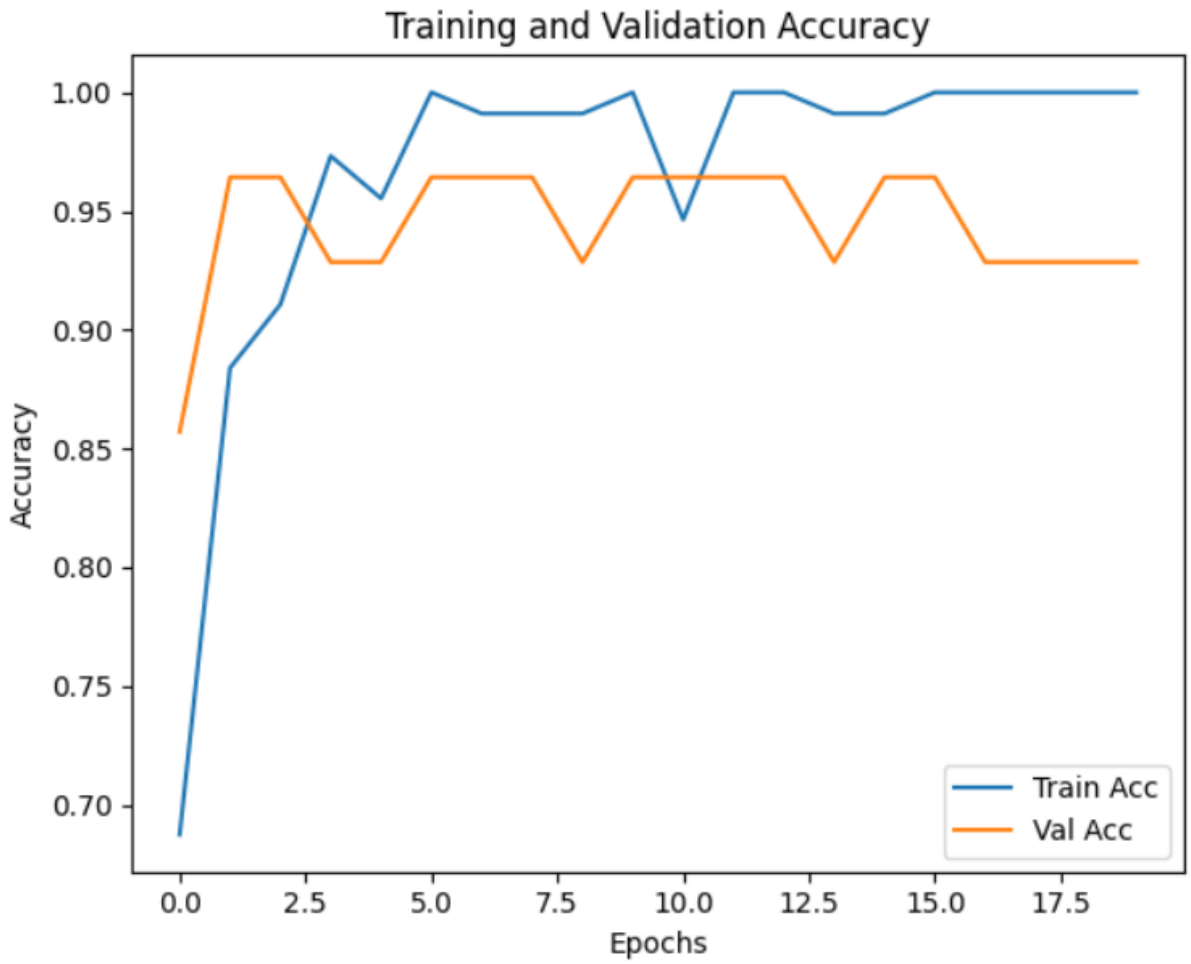
Our model achieved strong results on the test set, with the following metrics:

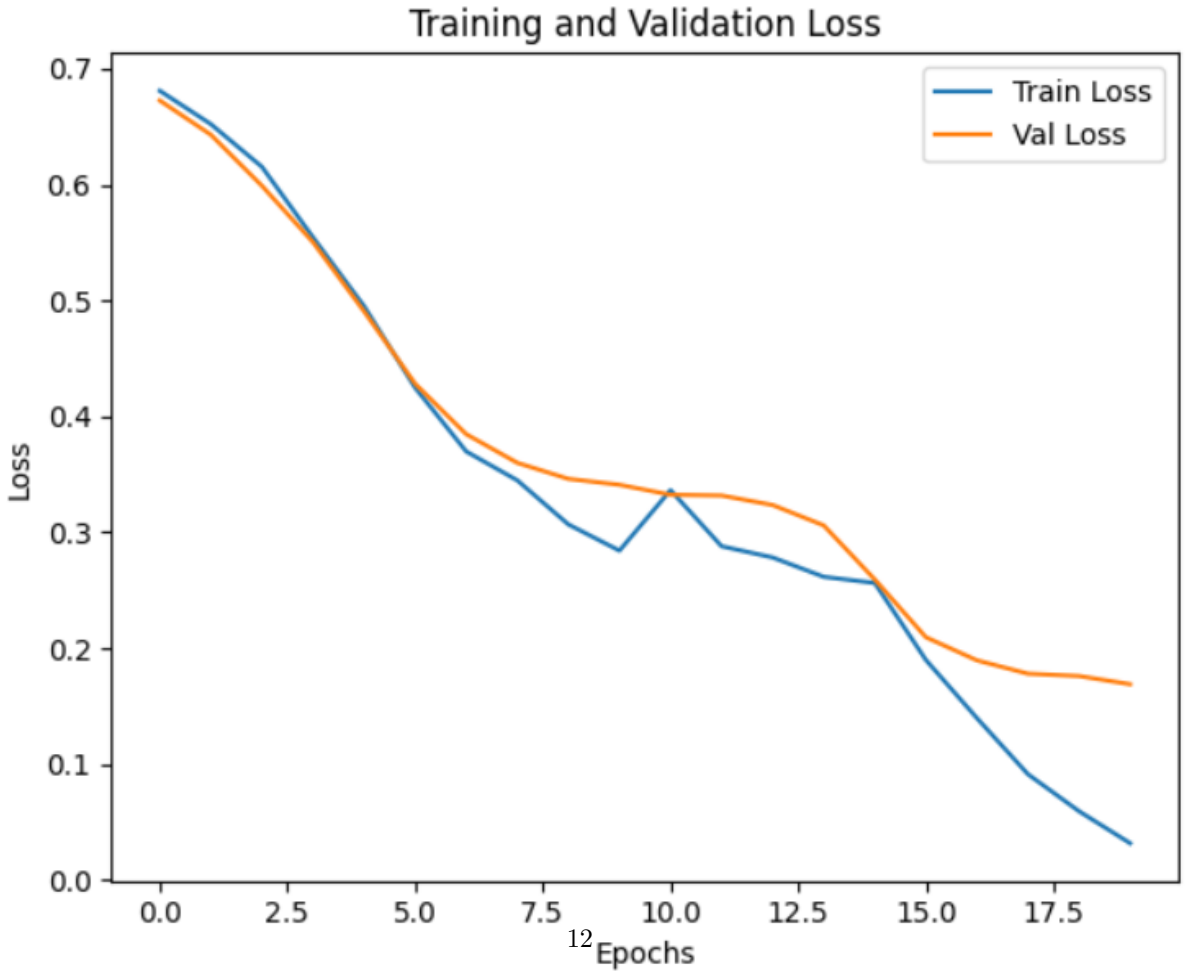
- Accuracy: 0.9286
- Precision: 1.0000
- Recall: 0.8571
- F1 Score: 0.9231
- AUC-ROC: 1.0000

This means our model correctly classified most videos, was perfect at avoiding false positives (precision 1.0), and caught most fakes (recall 0.8571). The AUC-ROC of 1.0 shows it separates real and fake videos well. Over 20 epochs, our training and validation performance improved steadily, as shown in our output (e.g., Epoch 20: Train Loss: 0.0317, Train Acc: 1.0000, Val Loss: 0.1689, Val Acc: 0.9286). We plotted these trends to visualize learning.

4.5.2 Misclassification Analysis

Our model misclassified 2 videos out of 28, both fake videos predicted as real:


Training and Validation Accuracy
Accuracy
Epochs
Train Acc
Val Acc
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Training and Validation Loss
Loss
Epochs
Train Loss
Val Loss
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

- **Example 1**: True Label: Fake, Predicted: Real (Real prob: 0.9184, Fake prob: 0.0816). This video likely had subtle manipulations, like smooth face swaps, that our features missed.

- **Example 2**: True Label: Fake, Predicted: Real (Real prob: 0.8733, Fake prob: 0.1267). Low frame quality or lighting may have confused the model.

Using `debug_test_predictions`, we found these videos had high-quality deepfakes or noisy frames, suggesting we could improve by tweaking our features, like adding audio cues or better landmark detection.

4.5.3 Attention Weight Visualization

To understand what our model focused on, we visualized attention weights from the attention mechanism (Section 3.4.2, Equation 4). High weights highlighted frames with suspicious movements, like unnatural blinks, confirming our model was looking at the right clues. This transparency, enabled by `visualize_attention`, was a key feature (Key Feature 3.3), making our results easier to trust and explain.

4.5.4 Single-Video Prediction

We tested our model on a single video, `W022_light_rightdown_surprise_camera_front.mp4`, using `predict_video`. The result was: *Prediction: Real (confidence: 0.6880), Real probability: 0.6880, Fake probability: 0.3120*. This shows our model can make practical predictions for new videos, a key feature (Key Feature 3.5), useful for real-world applications like checking social media clips. The moderate confidence suggests some frames had ambiguous features, but the correct prediction boosted our confidence in the model.

4.5.5 System Resource Utilization

We tracked how our system used Kaggle's resources with `print_system_stats`. Processing 70 real videos took 2 minutes 29 seconds, and 70 fake videos took 1 minute 55 seconds, showing fake videos were slightly shorter or simpler. Training for 20 epochs took about 30–40 minutes on a Kaggle GPU (P100), with each epoch taking 1–2 minutes. Memory usage stayed within Kaggle's 13 GB RAM limit, and disk outputs (e.g., `best_model.pth`, plots) used less than 1 GB. This efficiency made our system practical for a beginner project, running smoothly without crashes.

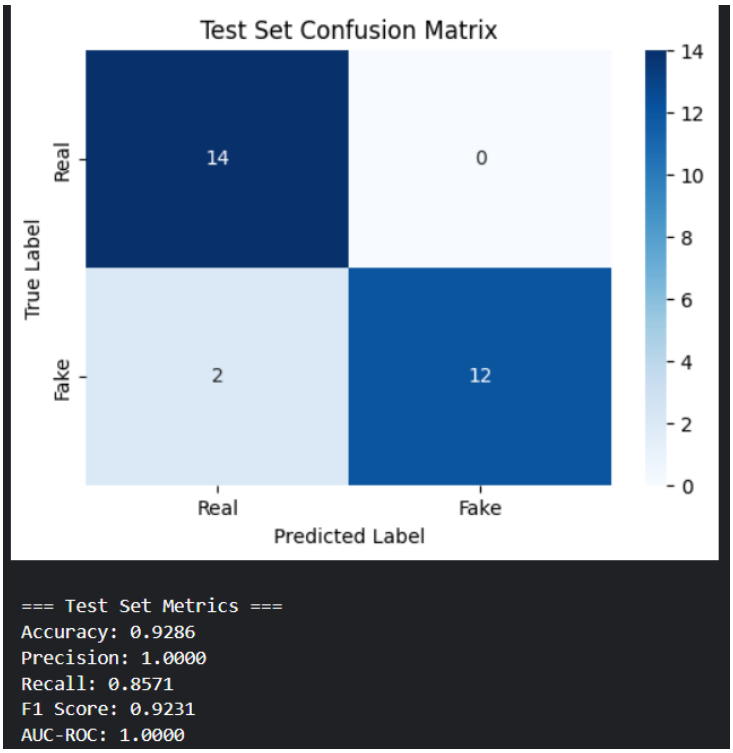


Figure 4: Confusion matrix for the test set, showing true vs. predicted labels for real and fake videos.

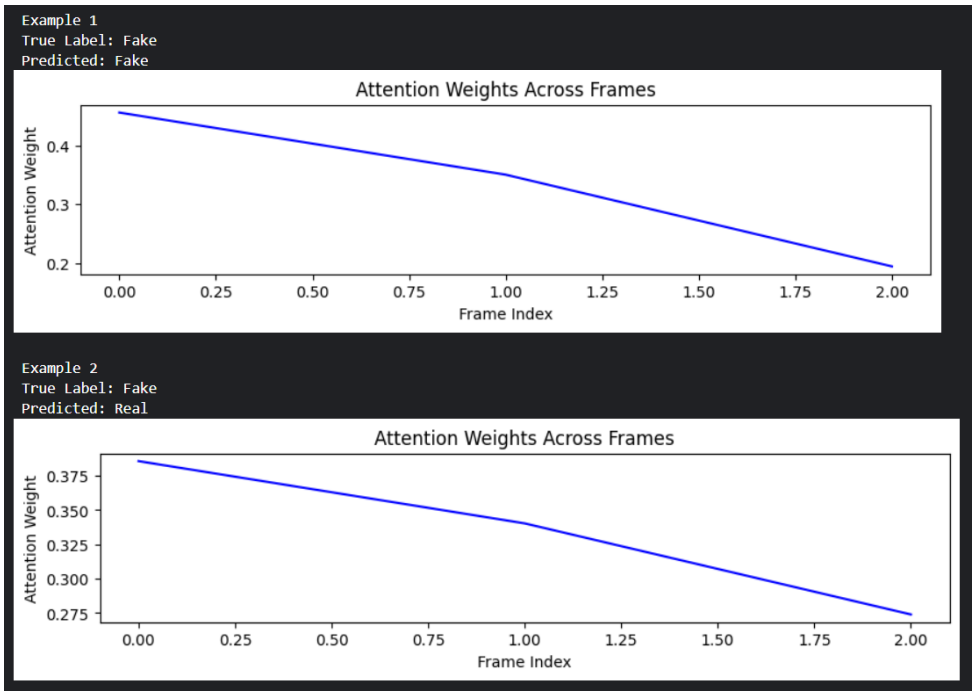


Figure 5: Attention weights for example test videos, showing which frames the model focused on to detect fakes.

5 Conclusion

We set out to confront the rising tide of deepfakes, videos that erode trust in the digital world, and we marveled at our creation—a system that empowers us to distinguish truth from deception. Built on Kaggle's accessible platform, our CNN-LSTM model, infused with an attention mechanism, unravels subtle clues through comprehensive feature extraction, from motion patterns to facial landmarks. Our data augmentation techniques stretched our small dataset, while dynamic visualizations of attention weights revealed our model's focus, making its decisions transparent even to beginners like us. The ability to predict a single video's authenticity brought our vision to life, offering a practical shield against misinformation. As George E. P. Box famously said, "All models are wrong, some are useful" [11]. This wisdom resonates with our journey: our model, though imperfect, is a useful ally, reliably spotting fakes and illuminating its reasoning. Its limitations, like occasional misclassifications, spur us to dream bigger—integrating audio features, training on vast datasets, or crafting real-time detection for social media and beyond. This project taught us that teamwork and deep learning can tackle urgent challenges, forging tools that are both powerful and approachable. We hope our work inspires fellow beginners to join this fight, building solutions that restore faith in what's real.

6 References

References

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, Cambridge, MA, USA, 2016.

[2] C. C. Aggarwal, *Neural Networks and Deep Learning: A Textbook*, Springer, Cham, Switzerland, 2018.

[3] R. Tolosana, R. Vera-Rodriguez, J. Fierrez, A. Morales, and J. Ortega-Garcia, *DeepFakes and Beyond: A Survey of Face Manipulation and Fake Detection*, Springer, Cham, Switzerland, 2020.

[4] M. Elpeltagy, A. Ismail, M. S. Zaki, and K. Eldahshan, "A novel smart deepfake video detection system," *Int. J. Adv. Comput. Sci. Appl.*, vol. 14, no. 1, pp. 1–9, 2023. [Online]. Available: https://www.researchgate.net/publication/368249556

[5] B. Yasser, J. Hani, S. M. Elgayar, and O. Abdelhameed, "Deepfake detection using EfficientNet and XceptionNet," in *Proc. 16th Int. Conf. Intell. Comput. Inf. Syst. (ICICIS)*, Jun. 2024, pp. 1–6, doi: 10.1109/ICICIS58388.2023.

[6] A. Rössler, D. Cozzolino, L. Verdoliva, C. Riess, J. Thies, and M. Nießner, "FaceForensics++: Learning to detect manipulated facial images," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1–11, doi: 10.1109/ICCV.2019.00009.

[7] A. Heidari, M. A. Ardeh, and M. J. Shayegan, "Deepfake detection using deep learning methods: A systematic and comprehensive review," *WIREs Data Mining Knowl. Discov.*, vol. 14, no. 3, p. e1520, May 2024, doi: 10.1002/widm.1520.

[8] G. E. P. Box, "Robustness in the strategy of scientific model building," in *Robustness in Statistics*, R. L. Launer and G. N. Wilkinson, Eds. New York, NY, USA: Academic Press, 1979, pp. 201–236.

[9] D. Afchar, V. Nozick, J. Yamagishi, and I. Echizen, "MesoNet: A compact facial video forgery detection network," in *Proc. IEEE Workshop Inf. Forensics Secur. (WIFS)*, Dec. 2018, pp. 1–7, doi: 10.1109/WIFS.2018.8630761.