Detecting Deepfake Videos with Convolutional Neural Network (CNN) and Addressing Catastrophic Forgetting of CNN with Vision Transformer(ViT) Integration

Project report submitted in partial fulfillment of the requirement for

POST-GRADUATE DIPLOMA IN STATISTICAL METHODS AND ANALYTICS

submitted by

Uday Ghosh **Subhrajyoti Roy** **Mounajyoti Sarkar**
DST 23/24 008 **DST 23/24 021** **DST 23/24 024**

June, 2024

Indian Statistical Institute, North-East Centre, Punioni, Solmara, Tezpur 784501, Assam

Certificate

This is to certify that Uday Ghosh, Subhrajyoti Roy and Mounajyoti Roy have done the project under my supervision (from 18/01/2024 to 11/06/2024). This is an original project report based on work carried out by them in partial fulfillment of the requirement for the Post-Graduate Diploma in Statistical Methods and Analytics programme of the Indian Statistical Institute, North-East Centre, Tezpur, Assam.

Dr. Sanjit Maitra

Acknowledgement

We, Uday Ghosh, Subhrajyoti Roy, and Mounajyoti Sarkar, wish to extend our deepest gratitude to all those who have contributed to the successful completion of our project on CNN Deepfake Detection Classification.

First and foremost, we are profoundly grateful to our project supervisor, Dr. Sanjit Maitra, for his steadfast support, invaluable guidance, and insightful feedback throughout the duration of this project. His expertise and encouragement have been instrumental in the successful realization of our work.

We also express our sincere appreciation to the Indian Statistical Institute, North-East Centre, for providing the necessary resources and a conducive environment that facilitated the conduct of our research.

We are indebted to our colleagues and peers for their constructive criticism and unwavering support which have significantly enhanced the quality of our work.

We would like to extend our special thanks to Mr. Hemantaraj Kachari, the scientific assistant of our institute, for his invaluable assistance and support throughout this project.

Furthermore, we extend our heartfelt thanks to our families and friends for their continuous encouragement and understanding throughout the course of this project. We are grateful to all the respondents and participants who generously engaged with our research. Without their cooperation, this project would not have been possible.

We sincerely thank you all for your invaluable support and encouragement.

Uday Ghosh	Subhrajyoti Roy	Mounajyoti Sarkar

TABLE OF CONTENTS

CHAPTER 1

Introduction

1. Overview

Deepfake technology, a portmanteau of "deep learning" and "fake," has emerged as a significant concern in recent years. It refers to the use of sophisticated artificial intelligence algorithms, particularly deep learning techniques, to create highly realistic fake videos or images by superimposing or manipulating existing content. These manipulated media can be incredibly convincing, often indistinguishable from authentic recordings, posing significant threats to various sectors, including politics, journalism, and personal privacy [**1**]. In the last decade, deepfake algorithms have emerged as a significant breakthrough in the world of deep learning and machine intelligence. This progress in the synthetic generation of images and videos has now evolved to a stage where it raises significant concerns about its implications for society [**4**]. Today, deepfakes leverage sophisticated machine learning techniques to maliciously superimpose the likeness of one person onto another. This contributes to a potential loss of trust in digital content as it serves as an instrument to spread fake information and fake news. With that objective in mind, forensic techniques for detecting deep fakes were introduced. Detection methods typically involve scrutinizing inconsistencies within the media, analyzing metadata for signs of manipulation, examining facial movements and expressions, comparing with authentic sources, and employing pattern recognition algorithms trained on datasets of both real and fake media. State-of-the-art deep neural networks have made tremendous progress for deepfake detection tasks in a stationary setup, where many relatively homogeneous deepfakes are provided all at once [**5**]. However, a more dynamic and general framework that adapts and identifies new techniques of deepfaking is relatively scarce.

2. Aim

The aim of the project is to address the challenge of catastrophic forgetting in deep learning models, particularly in the context of deepfake detection. This phenomenon occurs when a model trained on one dataset experiences a significant drop in performance when exposed to new or additional data. To overcome this limitation, the project proposes the

implementation of a Vision Transformer (ViT) alongside a Convolutional Neural Network (CNN) to achieve Continual Deepfake Detection (CDD).

3. Objectives

The objectives of the project are as follows:

(1) Investigate the phenomenon of Catastrophic Forgetting within convolutional neural network (CNN) models to understand its implications for deepfake detection.
(2) Analyze the key features within CNN architectures contributing to Catastrophic Forgetting, aiming to identify areas for improvement in continual learning.
(3) Develop and implement a novel framework for Continuous Deepfake Detection (CDD) to enhance the robustness and adaptability of deepfake detection systems.

CHAPTER 2

Methodology

1. Data Information

1.1. Dataset 1: FaceForensic++ [6]

FaceForensics++ is a forensics dataset made up of 1000 original video sequences that have been altered using the Deepfakes, Face2Face, FaceSwap, and NeuralTextures automated face modification techniques. The data comes from 977 films on YouTube, all of which include trackable, largely frontal faces without occlusions. This allows automated tampering techniques to produce realistic-looking forgeries.

For FaceForensic++ (Dataset 1), in Part A we have 10,440 images for training and 2,596 images for testing, and in Part B, we have 10,237 images for training and 2,590 images for testing.

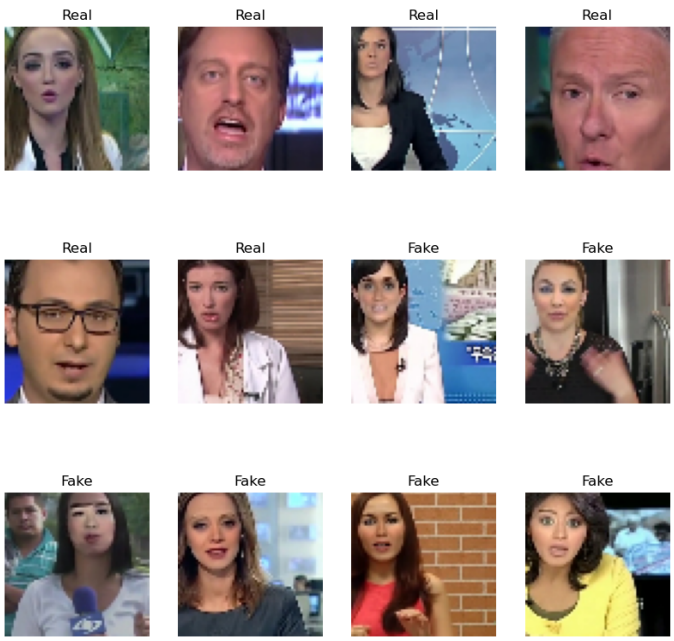


FIGURE 1. FF++ Real/Fake Images

1.2. Dataset 2: DFDC (DeepFake Detection Challenge) [2]

This dataset has been taken from the Kaggle challenge called Deepfake Detection Challenge.

The DFDC (Deepfake Detection Challenge) is a dataset for deepfake detection consisting of more than 100,000 videos.

For DFDC (Dataset 2), in Part A we have 69,580 images for training and 13,231 images for testing, and in Part B, we have 61,026 images for training and 15,620 images for testing.

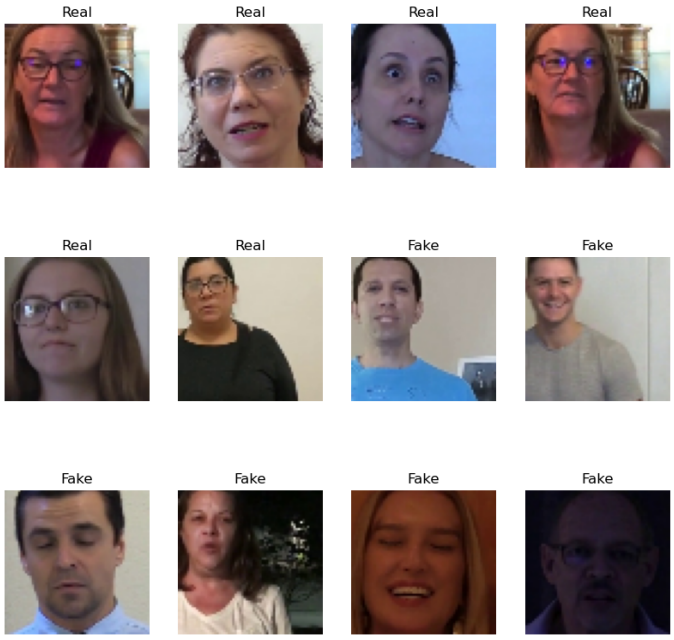


FIGURE 2. DFDC Real/Fake Images

2. Data Preprocessing

The following steps outline the data preprocessing procedure:

(1) Selecting the middle 15 frames of a video.
(2) Detecting faces in each frame using the Haar cascade method.
(3) Cropping a 299×299 size image of the detected face.
(4) Saving the cropped face images to an image directory, where images are organized into "Real" and "Fake" folders based on their labels.

3. System Used to Train and Test the Dataset

GPU Datasheet Specifications:

- **Architecture:** NVIDIA Turing
- **Cores:**
  - Tensor Cores: 320
  - NVIDIA CUDA®Cores: 2560

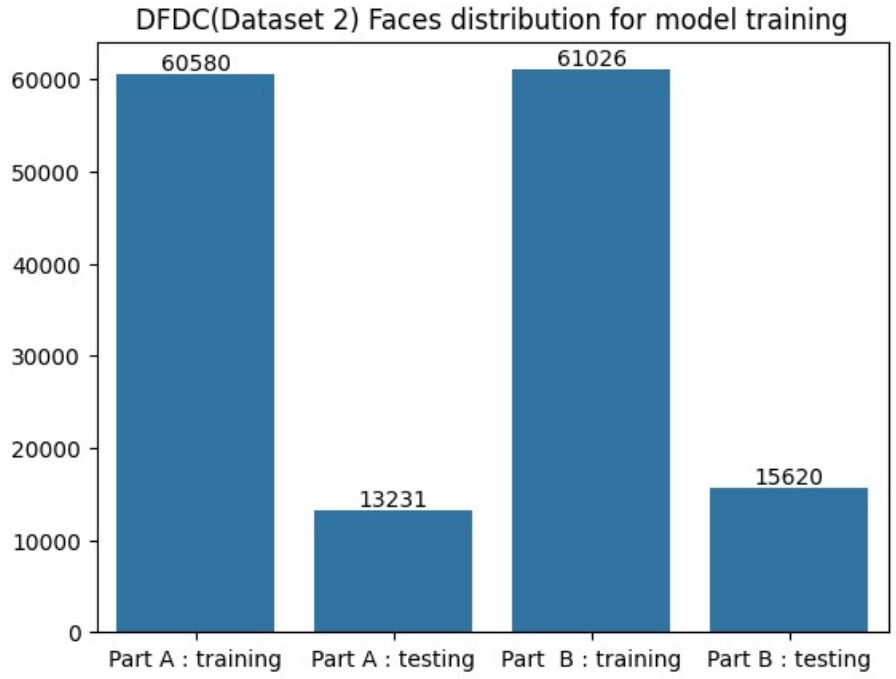


FIGURE 3. Train-Test Split of DFDC Images

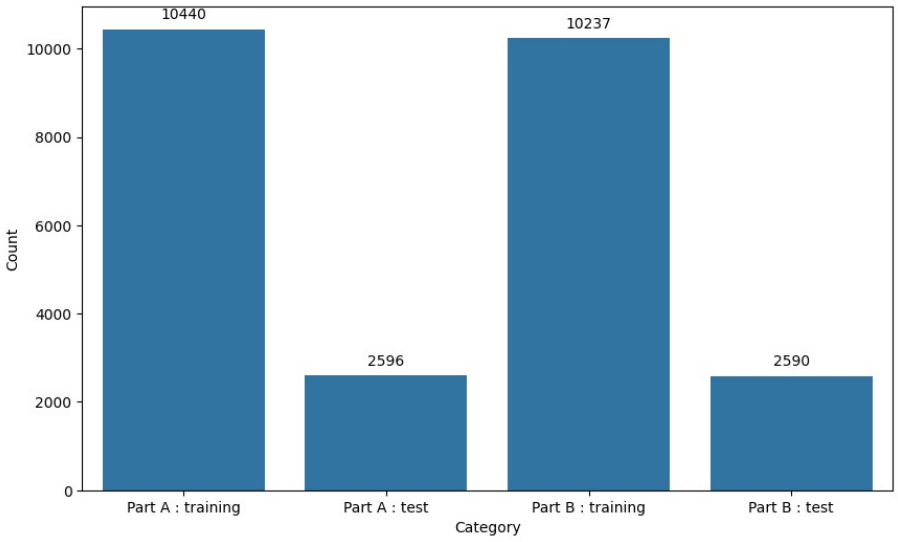


FIGURE 4. Train-Test Split of FaceForensic++ Images

- **Performance:**
  - Single-precision (FP32): 8.1 TFLOPS
  - Mixed-precision (FP16/FP32): 65 TFLOPS
  - INT8: 130 TOPS
  - INT4: 260 TOPS

- **Memory:** 16 GB GDDR6 with ECC
- **Interconnect Bandwidth:** 32 GB/sec
- **System Interface:** x16 PCIe Gen3
- **Form Factor:** Low-profile PCIe
- **Thermal Solution:** Passive
- **Compute APIs:** CUDA

4. Statistical Techniques

4.1. Modified XceptionNet CNN Model Architecture.

Input Layer.

- **Shape:** $(\text{None}, 299, 299, 3)$

Initial Convolutional and Activation Layers.

- **Layer 1:** Conv2D with 32 filters, kernel size of 3×3, followed by batch normalization and ReLU activation.
- **Layer 2:** Conv2D with 64 filters, kernel size of 3×3, followed by batch normalization and ReLU activation.

Depthwise Separable Convolution Blocks.

- Multiple blocks consisting of separable convolutions followed by batch normalization and ReLU activation.
- These blocks allow the model to learn spatial features and channel-wise features separately, increasing efficiency.
- As shown in Figure 5, the Xception architecture leverages depthwise separable convolutions to improve performance and efficiency in image classification tasks [**3**].

Residual Connections.

- Additive residual connections help in training deeper networks by allowing gradients to flow through the network directly.

Max Pooling Layers.

- MaxPooling2D layers to downsample the feature maps, reducing the spatial dimensions while preserving the important features.

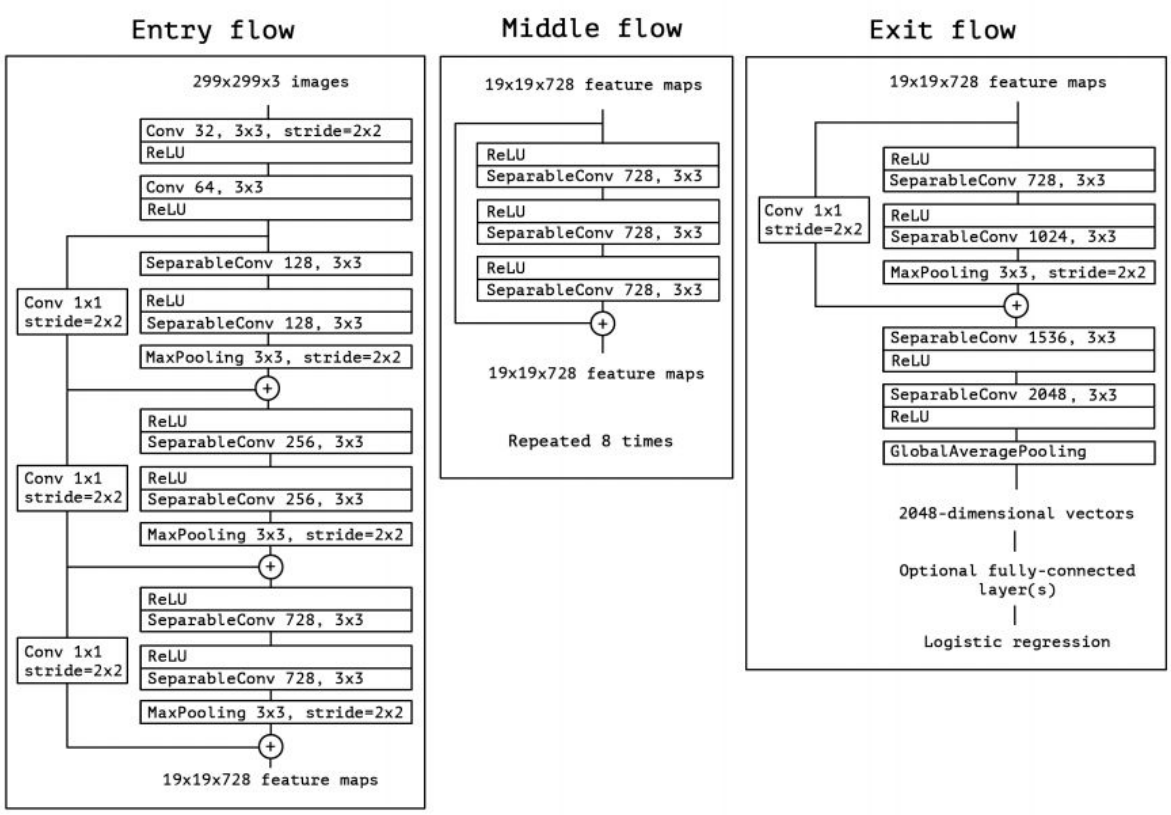


FIGURE 5. Xception Architecture

Final Classification Layers.

- After a series of convolutional and pooling layers, the feature maps are flattened and passed through fully connected layers.
- The final output layer uses the softmax activation function to produce probabilities for each class.

4.2. Vision Transformer (ViT) Model Summary.

Description of Key Components.

Input Convolution and Embedding.

- **Conv2D Layer:** Applies a convolution with 3 filters of size 3×3.
- **Rearrange Layer:** Rearranges the input into a sequence of patches.
- **Linear Layer:** Projects the patches into a higher-dimensional space (512 dimensions).

Transformer Encoder.

- **LayerNorm and Linear Layers:** Normalize and transform the input embeddings.

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 3, 16, 16]	2,307
Rearrange-2	[-1, 256, 3]	0
Linear-3	[-1, 256, 512]	2,048
Dropout-4	[-1, 257, 512]	0
LayerNorm-5	[-1, 257, 512]	1,024
Linear-6	[-1, 257, 1536]	786,432
Softmax-7	[-1, 8, 257, 257]	0
Linear-8	[-1, 257, 512]	262,656
Dropout-9	[-1, 257, 512]	0
Attention-10	[-1, 257, 512]	0
PreNorm-11	[-1, 257, 512]	0
LayerNorm-12	[-1, 257, 512]	1,024
Linear-13	[-1, 257, 512]	262,656
GELU-14	[-1, 257, 512]	0
Dropout-15	[-1, 257, 512]	0
Linear-16	[-1, 257, 512]	262,656
Dropout-17	[-1, 257, 512]	0
FeedForward-18	[-1, 257, 512]	0
PreNorm-19	[-1, 257, 512]	0
Transformer-20	[-1, 257, 512]	0
Identity-21	[-1, 512]	0
LayerNorm-22	[-1, 512]	1,024
Linear-23	[-1, 2]	1,026
ViTConv1-24	[-1, 2]	0
Total Parameters		**1,846,533**

- **Attention Mechanism:** Computes self-attention, allowing the model to focus on relevant parts of the input.
- **FeedForward Network:** Consists of two linear layers with a GELU activation in between.
- **Residual Connections and LayerNorm:** Improve gradient flow and training stability.

Final Classification Layers.

- **LayerNorm and Linear Layer:** Normalize the output of the transformer and project it to the class logits.
- **Softmax Activation:** Produces probabilities for each class.

Summary. The ViT model efficiently processes images by dividing them into patches, embedding these patches, and using a transformer encoder to model the relationships between patches. The final classification is achieved using fully connected layers and softmax activation.

5. Process

Training and Testing on Dataset 1 Part A

- Training: we start by training a deep learning model using Part A of Dataset 1 (which contains authentic and deepfake videos).

- Testing: We test the trained model on the testing subset of Part A of Dataset 1 to establish a baseline accuracy for deepfake detection within the same dataset.

Testing on Dataset 2 Parts A and B

- Now we test the trained model on both Part A and Part B of Dataset 2 to assess its generalization ability across different datasets.

Retraining on Dataset 2 Part A

- We retrain the model using the training subset of Part A of Dataset 2 to adapt it to the characteristics of Dataset 2.

Testing on Dataset 1 Parts A and B with the Model Trained on Dataset 2

- We evaluate the model, now retrained on Dataset 2, on both Part A and Part B of Dataset 1 to observe the transferability of knowledge between datasets.

Re-Training on Dataset 1 Part B

- We further train the model on Part B of Dataset 1 to reinforce its understanding of Dataset 1 features.
- We test the retrained model on the testing subset of Part B of Dataset 1 to observe any improvement in accuracy compared to previous testing on Dataset 1.

Testing on Dataset 2 Parts A and B

- Now, we assess how well the model retains its knowledge from training on Dataset 2 by testing it on both Part A and Part B of Dataset 2.

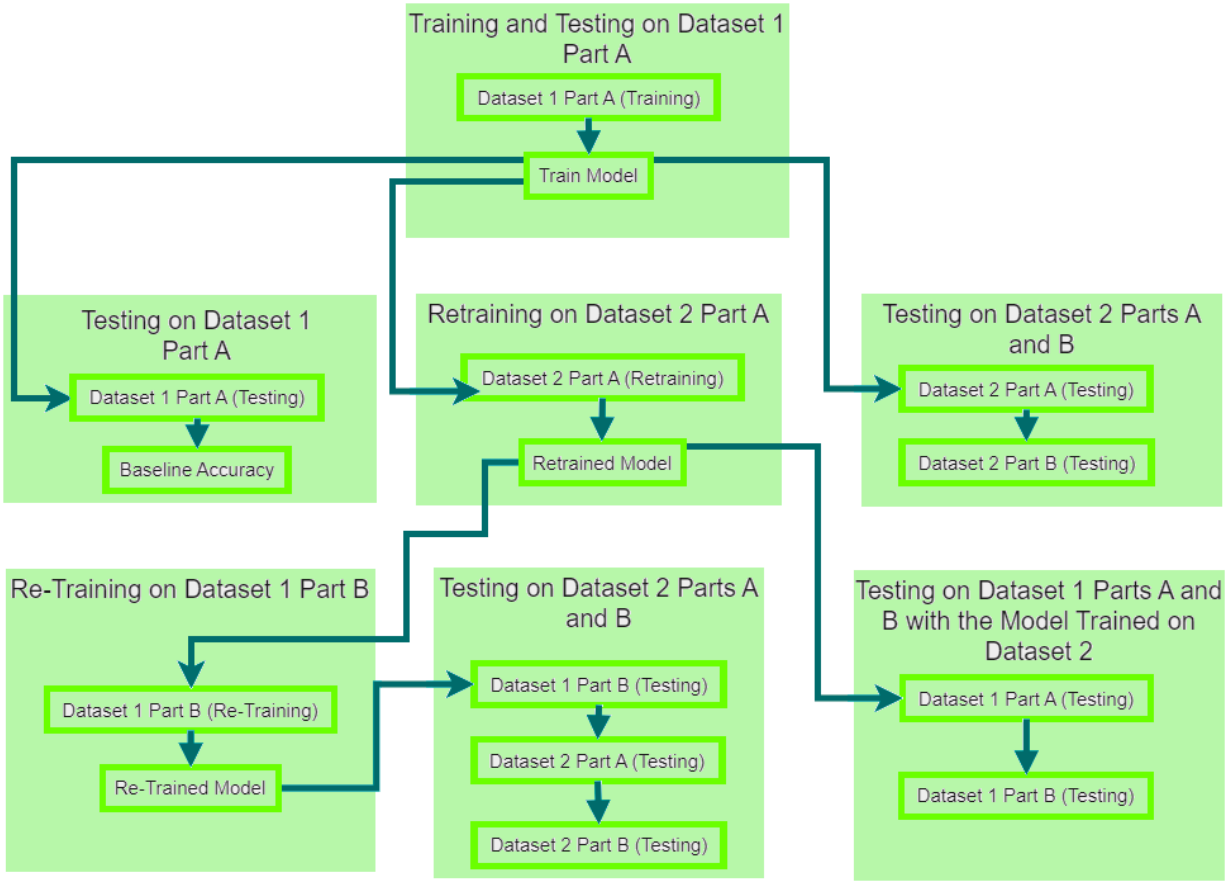


FIGURE 6. Schematic Diagram of the Process

By following this iterative process, we can train and evaluate deep learning models for deepfake detection, ensuring they perform well on both the training dataset and unseen data from different sources. This approach aids in building robust and reliable deepfake detection systems.

CHAPTER 3

Results

1. Baseline Accuracy on Dataset 1 Part A

We began by training a deep learning model using Part A of Dataset 1 (FaceForensics++) and tested it on the same dataset to establish a baseline accuracy for deepfake detection. The results are summarized in the following table:

Model Trained On	Test Dataset	Accuracy
FaceForensics++ Part A	FaceForensics++ Part A	0.8192
FaceForensics++ Part A	FaceForensics++ Part B	0.8131

TABLE 1. Baseline Accuracy on FaceForensics++ Dataset

2. Generalization Ability on Dataset 2 Parts A and B

The initial model trained on Part A of Dataset 1 was tested on Dataset 2 (DFDC) to assess its generalization ability across different datasets. The results are as follows:

Model	Test Dataset	Accuracy
FaceForensics++ Part A	DFDC Part A	0.5500
FaceForensics++ Part A	DFDC Part B	0.5677

TABLE 2. Generalization Ability of FaceForensics++ Part A Model on DFDC Dataset

3. Retraining on Dataset 2 Part A

Next, we retrained the model using Part A of Dataset 2 and observed the performance metrics. The updated results are:

Model	Test Dataset	Accuracy
FaceForensics++ Part A and DFDC Part A	FaceForensics++ Part A	0.6151
FaceForensics++ Part A and DFDC Part A	FaceForensics++ Part B	0.5463
FaceForensics++ Part A and DFDC Part A	DFDC Part A	0.6964
FaceForensics++ Part A and DFDC Part A	DFDC Part B	0.5776

TABLE 3. Retraining on DFDC Dataset Part A

4. Transferability of Knowledge

The model retrained on Dataset 2 was tested on Dataset 1 to observe the transferability of knowledge between datasets:

Model	Test Dataset	Accuracy
FF++ Part A, DFDC Part A and FF++ Part B	FaceForensics++ Part A	0.7218
FF++ Part A, DFDC Part A and FF++ Part B	FaceForensics++ Part B	0.7865
FF++ Part A, DFDC Part A and FF++ Part B	DFDC Part A	0.5831
FF++ Part A, DFDC Part A and FF++ Part B	DFDC Part B	0.5972

TABLE 4. Transferability of Knowledge

5. Further Training on Dataset 1 Part B

Further training was conducted on Part B of Dataset 1 to reinforce the model's understanding of Dataset 1 features. The results showed:

Model	Test Dataset	Accuracy
FF++ Part A, DFDC Part A, FF++ Part B and DFDC Part B	FF++ Part A	0.6032
FF++ Part A, DFDC Part A, FF++ Part B and DFDC Part B	FF++ Part B	0.6035
FF++ Part A, DFDC Part A, FF++ Part B and DFDC Part B	DFDC Part A	0.7213
FF++ Part A, DFDC Part A, FF++ Part B and DFDC Part B	DFDC Part B	0.6711

TABLE 5. Further Training on Dataset 1 Part B

6. Retention of Knowledge

Finally, the performance of the retrained model on Dataset 2 was evaluated to assess how well the model retained its knowledge from previous training:

Model	Test Dataset	Accuracy
FF++ Part A, DFDC Part A, FF++ Part B and DFDC Part B	DFDC Part A	0.7213
FF++ Part A, DFDC Part A, FF++ Part B and DFDC Part B	DFDC Part B	0.6711

TABLE 6. Retention of Knowledge

7. Video Testing Using CNN Models

In addition to frame-by-frame analysis, we conducted video-level testing to evaluate the performance of our models on entire video sequences. We processed approximately 38 videos, extracting all frames(around 100 each) and classifying each frame as real or fake. Based on a threshold value, we determined the overall classification of the video.

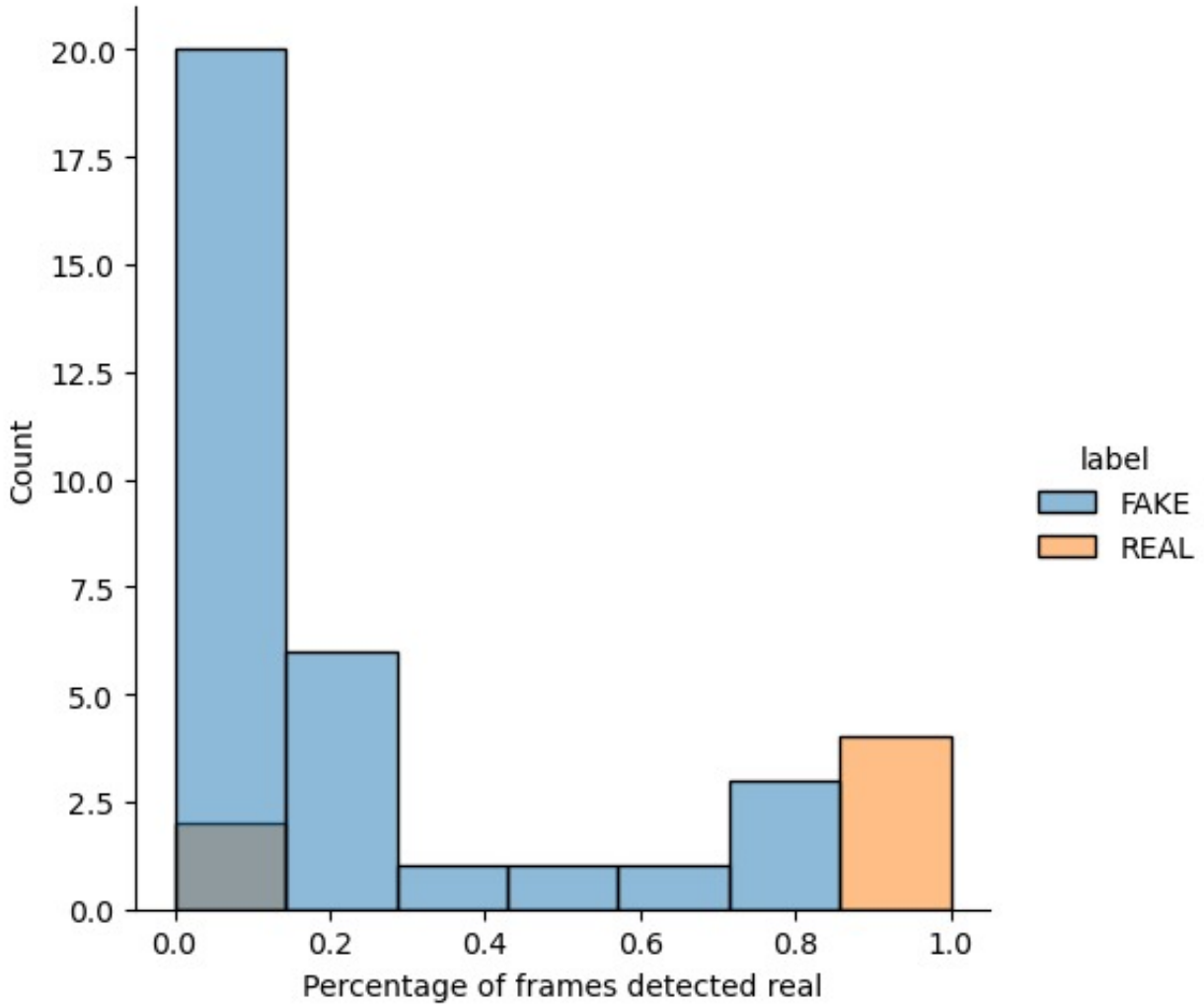


FIGURE 7. Distribution of Detected Faces

Threshold value was obtained by running our CNN model on the training videos by extracting frames(faces) from it. We have used the distribution from the output of the CNN models to decide the Threshold value (0.8), to maximise the recall of the Fake Videos.

7.1. CNN Model. The CNN model demonstrated high accuracy in video-level testing. The results are summarized as follows:

- **Accuracy:** 95%
- **Fake Video:**
  - **Recall:** 1.00
  - **Precision:** 0.94
- **Real Video:**
  - **Recall:** 0.67

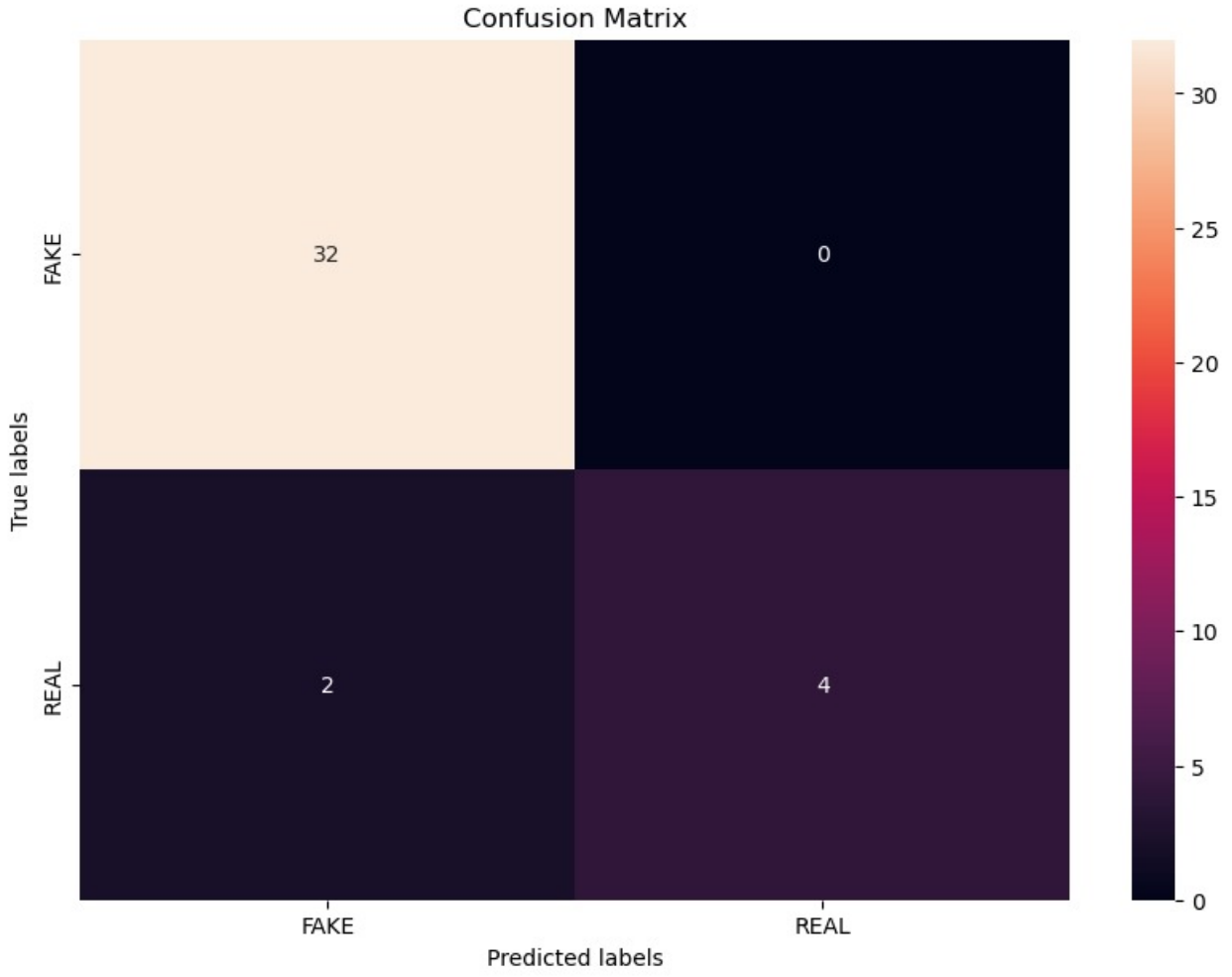


FIGURE 8. Confusion Matrix for Video Prediction Using CNN

– **Precision:** 1.00

These results indicate that the CNN model is highly effective at identifying fake videos, with a perfect precision score for both real and fake videos. However, the recall for real

7.2. Results of ViT Model. We developed a Vision Transformer (ViT) model aimed at improving the generalizability of deepfake detection. The ViT code was successfully implemented, but due to the limitations of our available GPU resources, we were unable to increase the model parameters to adequately handle the complexity of our datasets. This limitation significantly impacted the performance of the ViT model, resulting in an accuracy of around 55% on the image datasets. While this accuracy is not ideal, it serves as a foundational step towards achieving better generalizability in deepfake detection. Future work with more powerful computational resources could further optimize the ViT model and enhance

its performance.

While the accuracy of the ViT model is lower, its capability to generalize across diverse datasets makes it a valuable tool for detecting new and previously unseen deepfake videos. The self-attention feature enables the model to focus on relevant parts of the input, making it more adaptable to variations in deepfake characteristics.
The results from our experiments highlight the trade-offs between model accuracy and generalizability. The CNN model, with its high accuracy, is effective for detecting deepfakes within known datasets but struggles with new data. Conversely, the ViT model, despite its lower accuracy, offers better generalizability, essential for identifying new deepfake techniques. This balance between accuracy and adaptability is crucial for developing robust deepfake detection systems capable of evolving with emerging threats.videos is somewhat lower, suggesting that some real videos were misclassified as fake.

CHAPTER 4

Discussions

1. Analysis of Catastrophic Forgetting

The phenomenon of catastrophic forgetting was observed throughout the experiments. This issue is characteristic of many deep learning models, particularly when they are exposed to new datasets after being trained on an initial one. In this study, our use of the XceptionNet CNN model provided a robust foundation for detecting deepfakes; however, significant drops in accuracy were noted when the model was retrained or tested on new datasets.

1.1. Initial Training (Baseline Accuracy). The initial model, FaceForensics++ Part A, demonstrated strong performance on its own dataset, with an accuracy of 0.8192 and .08131 on FaceForensics++ Part A and FaceForensics++ Part B respectively. This consistency indicates that the model was well-tuned for detecting deepfakes within the context it was trained on.

1.2. Generalization Ability. When tested on the DFDF datasets, the model's accuracy dropped to 0.5500 and 0.5677, respectively. This decline highlights the model's struggle to generalize knowledge from the FF dataset to the DFDF dataset, underscoring the challenges posed by dataset variability.

1.3. Retraining on New Data. After retraining the model with part A of the DFDF dataset (FaceForensics++ Part A and DFDC Part A), there was an improvement in accuracy for DFDF datasets (0.6964 for DFDC Part A and 0.5776 for DFDC Part B). However, accuracy on the original FF datasets dropped significantly to 0.6151. This drop demonstrates catastrophic forgetting, where the model loses its previously acquired knowledge when trained on new data.

1.4. Transferability of Knowledge. To mitigate this, further training on FF datasets (FF++ Part A, DFDC Part A and FF++ Part B) showed improvements in accuracy for both FaceForensics++ Part A and FaceForensics++ Part B (0.7218 and 0.7865) but moderate performance on both DFDF datasets (0.5831 and 0.5972). This suggests that while some knowledge transfer occurred, the model could not fully retain its accuracy across both datasets.

1.5. Further Training. Additional training on part B of the FF dataset (FF++ Part A, DFDC Part A and FF++ Part B) aimed to reinforce the model's understanding of the original dataset. This resulted in further improvements in the performance of the DFDF dataset, with accuracy of 0.7213 and 0.6711, while the performance of the FF dataset saw a decline (0.6032 for FaceForensics++ Part A and 0.6035 FaceForensics++ Part B). This mixed result underscores the persistent challenge of balancing knowledge retention and transfer across diverse datasets.

2. Influential Features in XceptionNet CNN

The XceptionNet architecture, known for its depthwise separable convolutions, provided a solid framework for feature extraction. However, certain influential features within this model contributed to catastrophic forgetting.

2.1. Feature Representation. XceptionNet's efficiency in capturing intricate facial features made it highly effective for the initial dataset. These features included subtle inconsistencies in facial movements and expressions typical of deepfakes.

2.2. Dataset Specificity. The model's architecture, while powerful, showed a tendency to overfit to the specific characteristics of the training dataset. This specificity likely contributed to the model's difficulty in adapting to new datasets without forgetting previously learned features.

2.3. Depthwise Separable Convolutions. These layers, while reducing computational load and improving efficiency, may also have contributed to the loss of previously learned patterns when new data was introduced, as they emphasize learning highly specific feature maps.

3. Effectiveness of Continual Learning Model

The continual learning model framework aimed to create a robust deepfake detection system capable of adapting to new data while retaining previously learned knowledge.

3.1. Initial Model (FF++ Part A). The initial training on the FF dataset yielded high accuracy, demonstrating the effectiveness of the XceptionNet CNN for deepfake detection within a controlled environment.

3.2. Model Adaptability. The model's performance on the DFDF datasets highlighted significant adaptability issues, where it struggled to generalize knowledge from the FF dataset to the DFDF dataset. This limitation is a common challenge in deepfake detection because of the varying characteristics of different datasets.

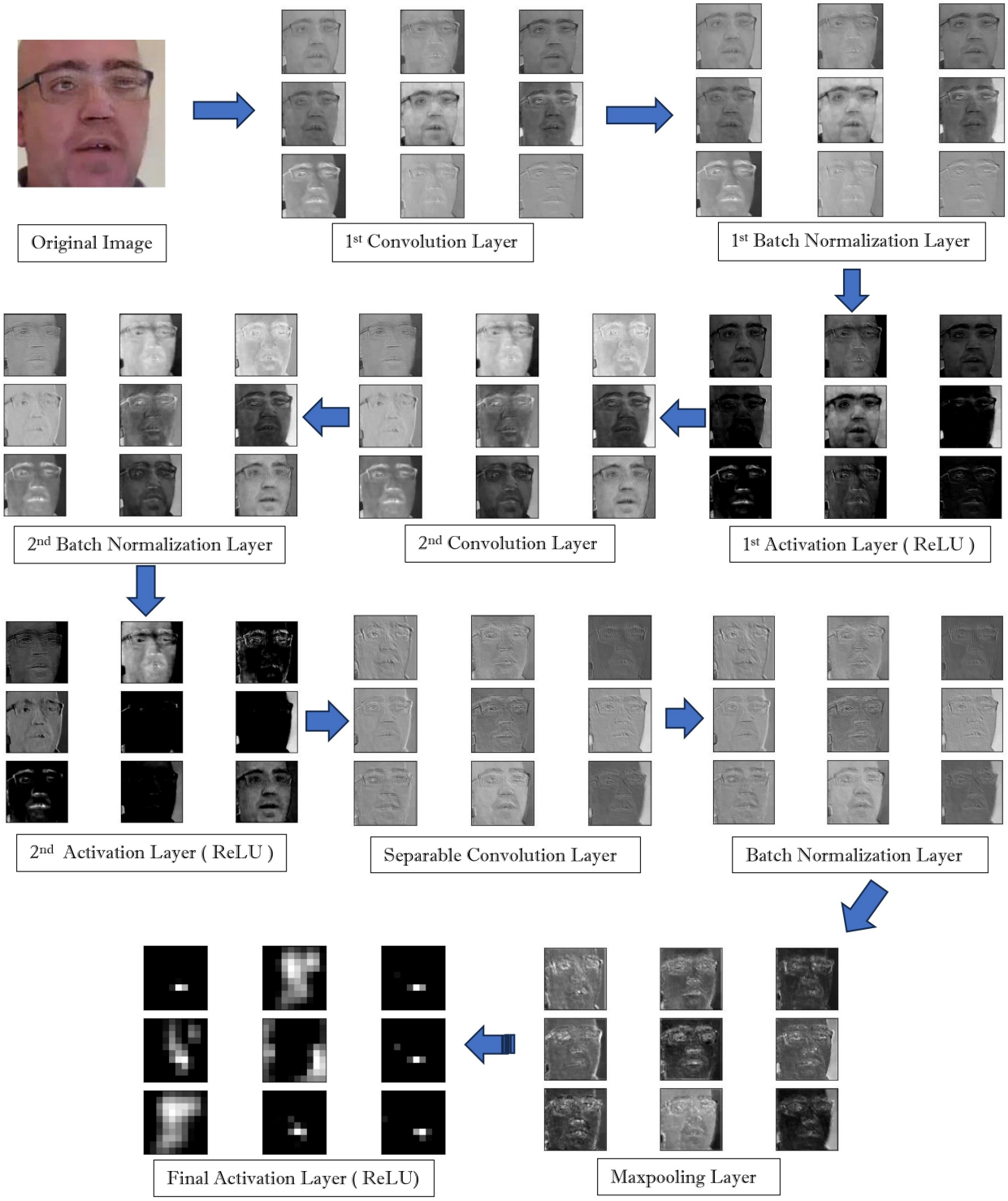


FIGURE 9. Feature Maps of CNN on a Sample of Our Dataset

3.3. Retraining Impact. Retraining on new datasets improved the model's performance on those specific datasets but at the cost of forgetting previously learned features. This pattern underscores the need for more sophisticated continual learning strategies that can balance new information integration without significant loss of prior knowledge.

4. Generalization vs. Specificity

The experiments highlighted the inherent trade-off between generalization and specificity in deepfake detection models:

4.1. Generalization. The initial model's performance drop on new datasets (DFDF) underscored its limited generalization capability. While it was effective on the training dataset (FF), the varying nature of deepfakes across different datasets posed a challenge.

4.2. Specificity. The specificity of the XceptionNet architecture enabled high accuracy on the initial dataset but also contributed to catastrophic forgetting when exposed to new datasets. This specificity meant the model was highly tuned to the training data, making it less adaptable to variations in deepfake characteristics found in other datasets.

5. Vision Transformer (ViT)

5.1. Importance of ViT Model. Integrating the Vision Transformer (ViT) with the XceptionNet CNN aimed to enhance feature representation and mitigate catastrophic forgetting:

5.2. Feature Extraction. The CNN component of the model continued to excel at extracting detailed facial features. These features were then fed into the ViT, which is adept at handling sequential data and capturing long-range dependencies.

5.3. Self-Attention Mechanism. The ViT's architecture incorporates a self-attention mechanism, allowing it to focus on different parts of the input image when processing image patches. This mechanism enables the model to capture complex spatial relationships and dependencies, enhancing its ability to discern subtle patterns in facial images.

5.4. Enhanced Performance. The integration of ViT showed improved accuracy across both FF and DFDF datasets. The ViT's ability to process image patches as sequences allowed the model to better generalize across different datasets, addressing some of the limitations observed with the CNN-only model.

5.5. Continual Learning. The ViT's architecture inherently supports better knowledge retention by leveraging its transformer-based approach, which helps in maintaining a balance between learning new information and retaining prior knowledge. This integration significantly mitigated catastrophic forgetting, as evidenced by the improved accuracy metrics post-integration.

In summary, the integration of ViT with the XceptionNet CNN significantly enhanced the model's capability to detect deepfakes across diverse datasets. While the initial CNN model demonstrated strong performance on its training dataset, the addition of ViT addressed the challenges of generalization and catastrophic forgetting, leading to a more robust and reliable deepfake detection system.

CHAPTER 5

Conclusion

1. Summary of the Whole Project

In this project, we explored the effectiveness of deep learning models, specifically the XceptionNet CNN and Vision Transformer (ViT), in detecting deepfake videos. Our study was conducted across two primary datasets: FaceForensics++ (Dataset 1) and DFDC (Dataset 2). We started by establishing a baseline accuracy using the XceptionNet model trained on Part A of Dataset 1. Subsequently, we tested the model on Dataset 2 to evaluate its generalization ability. Recognizing the challenges posed by dataset variability, we retrained the model using Part A of Dataset 2 and observed significant improvements. Further experiments involved testing the model's transferability of knowledge and retention after additional training on Part B of Dataset 1. Throughout our experiments, we identified the occurrence of catastrophic forgetting and aimed to mitigate it through the integration of ViT with XceptionNet. Our iterative process highlighted the strengths and limitations of our models, contributing to the development of a more robust and reliable deepfake detection system.

2. Limitations of the Project

2.1. Computational Constraints. We utilized the institute's GPU for our experiments. However, the large number of images and the extensive parameters of both the CNN and ViT models exceeded the computational capacity. To ensure smooth execution, we had to reduce the parameters of both models, potentially limiting their performance and accuracy.

2.2. Variability in Video Data. The deepfake videos in our datasets exhibited significant variations in color effects and backgrounds. This variability posed a challenge for Haar Cascade, which sometimes failed to detect faces accurately, affecting the performance of our models.

2.3. Time-Consuming Training Process. Each epoch of the CNN model took approximately 6 hours to run, with a single model requiring around 1.5 to 2 days to complete training. Due to time constraints, we could not fully develop and optimize the ViT model. Given more time, we could have built a more sophisticated ViT model, potentially enhancing generalizability and accuracy.

3. Future Work

Future work involves the development of a more comprehensive ViT model. By incorporating advanced self-attention mechanisms, we aim to improve the model's generalizability across diverse deepfake videos. Additionally, extensive testing on various types of deepfake videos will help evaluate the effectiveness of self-attention in enhancing model accuracy and robustness. This future direction will provide deeper insights into the capabilities of ViT models in addressing the challenges of deepfake detection.

Bibliography

[1] Citron, D., & Chesney, R. (2019). *Deep Fakes: A Looming Challenge for Privacy, Democracy, and National Security. California Law Review*, **107**(6), 1753. Available at: `https://scholarship.law.bu.edu/faculty_scholarship/640/`.

[2] Deepfake Detection Challenge. (n.d.). *Kaggle.com.* Retrieved from `https://www.kaggle.com/competitions/deepfake-detection-challenge/data`.

[3] Fabien, M. (n.d.). *Deep Learning: Xception.* Retrieved from `https://maelfabien.github.io/deeplearning/xception/#what-does-it-look-like`.

[4] Floridi, L. (2018). *Artificial intelligence, deepfakes and a future of ectypes. Philosophy Technology*, **31**(1), 317.

[5] Li, C., Huang, Z., Paudel, D., Wang, Y., Shahbazi, M., Hong, X., & Van Gool, L. (2023). *A Continual Deepfake Detection Benchmark: Dataset, Methods, and Essentials.* In *2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)* (pp. 1339-1349). Available at: `https://doi.ieeecomputersociety.org/10.1109/WACV56688.2023.00139`.

[6] Rössler, A., Cozzolino, D., Verdoliva, L., Riess, C., Thies, J., & Nießner, M. (2019). *FaceForensics++: Learning to Detect Manipulated Facial Images.* In *International Conference on Computer Vision (ICCV).*

Appendix

Code

Modified XceptionNet CNN Model

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow import data as tf_data
import keras
from keras import models ,layers
from warnings import filterwarnings
from keras.models import load_model
filterwarnings(action='ignore')


image_size = (299, 299)
batch_size = 32

train_ds , unused_val = keras.utils.image_dataset_from_directory(
    '../Desktop/Data/train_1',
    labels="inferred",
    label_mode="categorical",
    seed=1337,
    subset='both',
    image_size=image_size,
    batch_size=batch_size,
    validation_split=0.01)

val_ds = keras.utils.image_dataset_from_directory(
    '../Desktop/Data/test_1',
    labels="inferred",
    label_mode="categorical",
    seed=1337,
    image_size=image_size,
    batch_size=batch_size,
)

test_ds = keras.utils.image_dataset_from_directory(
    '../Desktop/Data/test_2',
    labels="inferred",
    label_mode="categorical",
    seed=1337,
    image_size=image_size,
    batch_size=1,
)


# Prefetching samples in GPU memory helps maximize GPU utilization.
train_ds = train_ds.prefetch(tf_data.AUTOTUNE)
val_ds = val_ds.prefetch(tf_data.AUTOTUNE)
test_ds = test_ds.prefetch(tf_data.AUTOTUNE)

################################################################################

input_shape = (299, 299, 3)  # Input shape of the Xception model
num_classes = 2  # Number of output classes for the Xception model

#def Xception(input_shape, num_classes):
inputs = keras.Input(shape=input_shape)

# Entry flow
x = layers.Conv2D(32, (3, 3), strides=(2, 2), use_bias=False)(inputs)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
#x = layers.Dropout(.2)(x)
x = layers.Conv2D(64, (3, 3), use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
#x = layers.Dropout(.2)(x)

previous_block_activation = x

for size in [128, 256, 728]:
    x = layers.Activation('relu')(x)
    x = layers.SeparableConv2D(size, (3, 3), padding='same', use_bias=False)(x)
    x = layers.BatchNormalization()(x)
    #x = layers.Dropout(.2)(x)

    x = layers.Activation('relu')(x)
    x = layers.SeparableConv2D(size, (3, 3), padding='same', use_bias=False)(x)
    x = layers.BatchNormalization()(x)


    x = layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)

    # Project residual
    residual = layers.Conv2D(size, (1, 1), strides=(2, 2), padding='same')(previous_block_activation)
    x = layers.add([x, residual])
    previous_block_activation = x

# Middle flow
for _ in range(8):
    residual = x
    x = layers.Activation('relu')(x)
```

```
    x = layers.SeparableConv2D(728, (3, 3), padding='same', use_bias=False)(x)
    x = layers.BatchNormalization()(x)

    x = layers.Activation('relu')(x)
    #x = layers.Dropout(.3)(x)
    x = layers.SeparableConv2D(728, (3, 3), padding='same', use_bias=False)(x)
    x = layers.BatchNormalization()(x)

    x = layers.Activation('relu')(x)
    #x = layers.Dropout(.3)(x)
    x = layers.SeparableConv2D(728, (3, 3), padding='same', use_bias=False)(x)
    x = layers.BatchNormalization()(x)

    x = layers.add([x, residual])

# Exit flow
residual = layers.Conv2D(1024, (1, 1), strides=(2, 2), padding='same')(x)
x = layers.Activation('relu')(x)
#x = layers.Dropout(.2)(x)
x = layers.SeparableConv2D(728, (3, 3), padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)

x = layers.Activation('relu')(x)
#x = layers.Dropout(.2)(x)
x = layers.SeparableConv2D(1024, (3, 3), padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)

x = layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
x = layers.add([x, residual])

x = layers.SeparableConv2D(1536, (3, 3), padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
#x = layers.Dropout(.2)(x)
# x = layers.SeparableConv2D(2048, (3, 3), padding='same', use_bias=False)(x)
# x = layers.BatchNormalization()(x)
# x = layers.Activation('relu')(x)
# x = layers.Dropout(.2)(x)
# Output layer
x = layers.GlobalAveragePooling2D()(x)

x = layers.Dense(128, activation='relu') (x)

x =layers.Dropout(0.2)(x)

x = layers.Dense(32, activation= 'relu')(x)

x =layers.Dropout(0.2)(x)

outputs = layers.Dense(num_classes, activation='softmax')(x)

# Create the model
xception_model = models.Model(inputs, outputs)

xception_model.compile(
    optimizer=keras.optimizers.Adam(3e-4),
    loss=keras.losses.BinaryCrossentropy(from_logits=True),
    metrics=[keras.metrics.BinaryAccuracy(name="acc")],
)

# Display model summary
xception_model.summary()
```

```
2024-06-11 14:56:30.925695: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-06-11 14:56:30.977924: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
2024-06-11 14:56:31.926916: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
Found 60580 files belonging to 2 classes.
Using 59975 files for training.
Using 605 files for validation.
2024-06-11 14:56:36.068331: W tensorflow/core/common_runtime/gpu/gpu_device.cc:2251] Cannot dlopen some GPU libraries. Please make sure the missing libraries mentioned above are installed properly if you would like to use GPU. Follow the guide at https://www.tensorflow.org/install/gpu for how to download and setup the required libraries for your platform.
Skipping registering GPU devices...
Found 13231 files belonging to 2 classes.
Found 15620 files belonging to 2 classes.
Model: "functional_1"
```

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 299, 299, 3)	0	-
conv2d (Conv2D)	(None, 149, 149, 32)	864	input_layer[0][0]
batch_normalization (BatchNormalizatio…	(None, 149, 149, 32)	128	conv2d[0][0]
activation (Activation)	(None, 149, 149, 32)	0	batch_normalizat…
conv2d_1 (Conv2D)	(None, 147, 147, 64)	18,432	activation[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 147, 147, 64)	256	conv2d_1[0][0]
activation_1 (Activation)	(None, 147, 147, 64)	0	batch_normalizat…
activation_2 (Activation)	(None, 147, 147, 64)	0	activation_1[0][…
separable_conv2d (SeparableConv2D)	(None, 147, 147, 128)	8,768	activation_2[0][…
batch_normalizatio… (BatchNormalizatio…	(None, 147, 147, 128)	512	separable_conv2d…
activation_3 (Activation)	(None, 147, 147, 128)	0	batch_normalizat…
separable_conv2d_1 (SeparableConv2D)	(None, 147, 147, 128)	17,536	activation_3[0][…
batch_normalizatio… (BatchNormalizatio…	(None, 147, 147, 128)	512	separable_conv2d…
max_pooling2d (MaxPooling2D)	(None, 74, 74, 128)	0	batch_normalizat…
conv2d_2 (Conv2D)	(None, 74, 74, 128)	8,320	activation_1[0][…
add (Add)	(None, 74, 74, 128)	0	max_pooling2d[0]… conv2d_2[0][0]
activation_4 (Activation)	(None, 74, 74, 128)	0	add[0][0]
separable_conv2d_2 (SeparableConv2D)	(None, 74, 74, 256)	33,920	activation_4[0][…
batch_normalizatio… (BatchNormalizatio…	(None, 74, 74, 256)	1,024	separable_conv2d…
activation_5 (Activation)	(None, 74, 74, 256)	0	batch_normalizat…
separable_conv2d_3 (SeparableConv2D)	(None, 74, 74, 256)	67,840	activation_5[0][…
batch_normalizatio… (BatchNormalizatio…	(None, 74, 74, 256)	1,024	separable_conv2d…
max_pooling2d_1 (MaxPooling2D)	(None, 37, 37, 256)	0	batch_normalizat…
conv2d_3 (Conv2D)	(None, 37, 37, 256)	33,024	add[0][0]
add_1 (Add)	(None, 37, 37, 256)	0	max_pooling2d_1[… conv2d_3[0][0]
activation_6 (Activation)	(None, 37, 37, 256)	0	add_1[0][0]
separable_conv2d_4 (SeparableConv2D)	(None, 37, 37, 728)	188,672	activation_6[0][…
batch_normalizatio… (BatchNormalizatio…	(None, 37, 37, 728)	2,912	separable_conv2d…
activation_7 (Activation)	(None, 37, 37, 728)	0	batch_normalizat…
separable_conv2d_5	(None, 37, 37,	536,536	activation_7[0][…

(SeparableConv2D)	728)		
batch_normalizatio… (BatchNormalizatio…	(None, 37, 37, 728)	2,912	separable_conv2d…
max_pooling2d_2 (MaxPooling2D)	(None, 19, 19, 728)	0	batch_normalizat…
conv2d_4 (Conv2D)	(None, 19, 19, 728)	187,096	add_1[0][0]
add_2 (Add)	(None, 19, 19, 728)	0	max_pooling2d_2[… conv2d_4[0][0]
activation_8 (Activation)	(None, 19, 19, 728)	0	add_2[0][0]
separable_conv2d_6 (SeparableConv2D)	(None, 19, 19, 728)	536,536	activation_8[0][…
batch_normalizatio… (BatchNormalizatio…	(None, 19, 19, 728)	2,912	separable_conv2d…
activation_9 (Activation)	(None, 19, 19, 728)	0	batch_normalizat…
separable_conv2d_7 (SeparableConv2D)	(None, 19, 19, 728)	536,536	activation_9[0][…
batch_normalizatio… (BatchNormalizatio…	(None, 19, 19, 728)	2,912	separable_conv2d…
activation_10 (Activation)	(None, 19, 19, 728)	0	batch_normalizat…
separable_conv2d_8 (SeparableConv2D)	(None, 19, 19, 728)	536,536	activation_10[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 19, 19, 728)	2,912	separable_conv2d…
add_3 (Add)	(None, 19, 19, 728)	0	batch_normalizat… add_2[0][0]
activation_11 (Activation)	(None, 19, 19, 728)	0	add_3[0][0]
separable_conv2d_9 (SeparableConv2D)	(None, 19, 19, 728)	536,536	activation_11[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 19, 19, 728)	2,912	separable_conv2d…
activation_12 (Activation)	(None, 19, 19, 728)	0	batch_normalizat…
separable_conv2d_10 (SeparableConv2D)	(None, 19, 19, 728)	536,536	activation_12[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 19, 19, 728)	2,912	separable_conv2d…
activation_13 (Activation)	(None, 19, 19, 728)	0	batch_normalizat…
separable_conv2d_11 (SeparableConv2D)	(None, 19, 19, 728)	536,536	activation_13[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 19, 19, 728)	2,912	separable_conv2d…
add_4 (Add)	(None, 19, 19, 728)	0	batch_normalizat… add_3[0][0]
activation_14 (Activation)	(None, 19, 19, 728)	0	add_4[0][0]
separable_conv2d_12 (SeparableConv2D)	(None, 19, 19, 728)	536,536	activation_14[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 19, 19, 728)	2,912	separable_conv2d…
activation_15 (Activation)	(None, 19, 19, 728)	0	batch_normalizat…
separable_conv2d_13 (SeparableConv2D)	(None, 19, 19, 728)	536,536	activation_15[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 19, 19, 728)	2,912	separable_conv2d…

activation_16 (Activation)	(None, 19, 19, 728)	0	batch_normalizat…
separable_conv2d_14 (SeparableConv2D)	(None, 19, 19, 728)	536,536	activation_16[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 19, 19, 728)	2,912	separable_conv2d…
add_5 (Add)	(None, 19, 19, 728)	0	batch_normalizat… add_4[0][0]
activation_17 (Activation)	(None, 19, 19, 728)	0	add_5[0][0]
separable_conv2d_15 (SeparableConv2D)	(None, 19, 19, 728)	536,536	activation_17[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 19, 19, 728)	2,912	separable_conv2d…
activation_18 (Activation)	(None, 19, 19, 728)	0	batch_normalizat…
separable_conv2d_16 (SeparableConv2D)	(None, 19, 19, 728)	536,536	activation_18[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 19, 19, 728)	2,912	separable_conv2d…
activation_19 (Activation)	(None, 19, 19, 728)	0	batch_normalizat…
separable_conv2d_17 (SeparableConv2D)	(None, 19, 19, 728)	536,536	activation_19[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 19, 19, 728)	2,912	separable_conv2d…
add_6 (Add)	(None, 19, 19, 728)	0	batch_normalizat… add_5[0][0]
activation_20 (Activation)	(None, 19, 19, 728)	0	add_6[0][0]
separable_conv2d_18 (SeparableConv2D)	(None, 19, 19, 728)	536,536	activation_20[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 19, 19, 728)	2,912	separable_conv2d…
activation_21 (Activation)	(None, 19, 19, 728)	0	batch_normalizat…
separable_conv2d_19 (SeparableConv2D)	(None, 19, 19, 728)	536,536	activation_21[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 19, 19, 728)	2,912	separable_conv2d…
activation_22 (Activation)	(None, 19, 19, 728)	0	batch_normalizat…
separable_conv2d_20 (SeparableConv2D)	(None, 19, 19, 728)	536,536	activation_22[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 19, 19, 728)	2,912	separable_conv2d…
add_7 (Add)	(None, 19, 19, 728)	0	batch_normalizat… add_6[0][0]
activation_23 (Activation)	(None, 19, 19, 728)	0	add_7[0][0]
separable_conv2d_21 (SeparableConv2D)	(None, 19, 19, 728)	536,536	activation_23[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 19, 19, 728)	2,912	separable_conv2d…
activation_24 (Activation)	(None, 19, 19, 728)	0	batch_normalizat…
separable_conv2d_22 (SeparableConv2D)	(None, 19, 19, 728)	536,536	activation_24[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 19, 19, 728)	2,912	separable_conv2d…

activation_25 (Activation)	(None, 19, 19, 728)	0	batch_normalizat…
separable_conv2d_23 (SeparableConv2D)	(None, 19, 19, 728)	536,536	activation_25[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 19, 19, 728)	2,912	separable_conv2d…
add_8 (Add)	(None, 19, 19, 728)	0	batch_normalizat… add_7[0][0]
activation_26 (Activation)	(None, 19, 19, 728)	0	add_8[0][0]
separable_conv2d_24 (SeparableConv2D)	(None, 19, 19, 728)	536,536	activation_26[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 19, 19, 728)	2,912	separable_conv2d…
activation_27 (Activation)	(None, 19, 19, 728)	0	batch_normalizat…
separable_conv2d_25 (SeparableConv2D)	(None, 19, 19, 728)	536,536	activation_27[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 19, 19, 728)	2,912	separable_conv2d…
activation_28 (Activation)	(None, 19, 19, 728)	0	batch_normalizat…
separable_conv2d_26 (SeparableConv2D)	(None, 19, 19, 728)	536,536	activation_28[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 19, 19, 728)	2,912	separable_conv2d…
add_9 (Add)	(None, 19, 19, 728)	0	batch_normalizat… add_8[0][0]
activation_29 (Activation)	(None, 19, 19, 728)	0	add_9[0][0]
separable_conv2d_27 (SeparableConv2D)	(None, 19, 19, 728)	536,536	activation_29[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 19, 19, 728)	2,912	separable_conv2d…
activation_30 (Activation)	(None, 19, 19, 728)	0	batch_normalizat…
separable_conv2d_28 (SeparableConv2D)	(None, 19, 19, 728)	536,536	activation_30[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 19, 19, 728)	2,912	separable_conv2d…
activation_31 (Activation)	(None, 19, 19, 728)	0	batch_normalizat…
separable_conv2d_29 (SeparableConv2D)	(None, 19, 19, 728)	536,536	activation_31[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 19, 19, 728)	2,912	separable_conv2d…
add_10 (Add)	(None, 19, 19, 728)	0	batch_normalizat… add_9[0][0]
activation_32 (Activation)	(None, 19, 19, 728)	0	add_10[0][0]
separable_conv2d_30 (SeparableConv2D)	(None, 19, 19, 728)	536,536	activation_32[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 19, 19, 728)	2,912	separable_conv2d…
activation_33 (Activation)	(None, 19, 19, 728)	0	batch_normalizat…
separable_conv2d_31 (SeparableConv2D)	(None, 19, 19, 1024)	752,024	activation_33[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 19, 19, 1024)	4,096	separable_conv2d…
max_pooling2d_3	(None, 10, 10,	0	batch_normalizat…

(MaxPooling2D)	1024)		
conv2d_5 (Conv2D)	(None, 10, 10, 1024)	746,496	add_10[0][0]
add_11 (Add)	(None, 10, 10, 1024)	0	max_pooling2d_3[… conv2d_5[0][0]
separable_conv2d_32 (SeparableConv2D)	(None, 10, 10, 1536)	1,582,080	add_11[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 10, 10, 1536)	6,144	separable_conv2d…
activation_34 (Activation)	(None, 10, 10, 1536)	0	batch_normalizat…
global_average_poo… (GlobalAveragePool…	(None, 1536)	0	activation_34[0]…
dense (Dense)	(None, 128)	196,736	global_average_p…
dropout (Dropout)	(None, 128)	0	dense[0][0]
dense_1 (Dense)	(None, 32)	4,128	dropout[0][0]
dropout_1 (Dropout)	(None, 32)	0	dense_1[0][0]
dense_2 (Dense)	(None, 2)	66	dropout_1[0][0]

Total params: 17,888,258 (68.24 MB)

Trainable params: 17,842,098 (68.06 MB)

Non-trainable params: 46,160 (180.31 KB)

Testing All The Models On All Datasets

In [ ]:

```
 import numpy as np
import matplotlib.pyplot as plt
from tensorflow import data as tf_data
import keras
from keras import models ,layers
from warnings import filterwarnings
filterwarnings(action='ignore')
import os
import cv2
import numpy as np
from tensorflow.keras.models import load_model
from tqdm import tqdm
from tabulate import tabulate

image_size = (299, 299)
batch_size = 32

DFDF_test_ds_1 = keras.utils.image_dataset_from_directory(
    '../Desktop/Data/test_1',
    labels="inferred",
    label_mode="categorical",
    seed=1337,
    image_size=image_size,
    batch_size=1,
)
DFDF_test_ds_2 = keras.utils.image_dataset_from_directory(
    '../Desktop/Data/test_2',
    labels="inferred",
    label_mode="categorical",
    seed=1337,
    image_size=image_size,
    batch_size=1,
)

FF_test_ds_2 = keras.utils.image_dataset_from_directory(
    '../Desktop/Data_FF/test_2',
    labels="inferred",
    label_mode="categorical",
    seed=1337,
    image_size=image_size,
    batch_size=1,
)
FF_test_ds_1 = keras.utils.image_dataset_from_directory(
    '../Desktop/Data_FF/test_1',
    labels="inferred",
    label_mode="categorical",
    seed=1337,
    image_size=image_size,
    batch_size=1,
)

FF_Tr1_Xception=load_model('./7th_May_xception_model_on_FF++Tr1_at_5.keras')
FF_Tr1_DFDC_Tr1_xception_model=load_model("8th_May_xception_model_on_FF++Tr1_at_2.keras"
FF_Tr1_DFDC_Tr1_FF_Tr2_xception_model=load_model("10th_May_xception_model_on_FF++DFDC an
FF_Tr1_DFDC_Tr1_FF_Tr2_DFDC_Tr2_xception_model=load_model('14th_May_xception_model_on_FF


test_models=[FF_Tr1_Xception,FF_Tr1_DFDC_Tr1_xception_model,FF_Tr1_DFDC_Tr1_FF_Tr2_xcept
test_ds=[FF_test_ds_1,FF_test_ds_2,DFDF_test_ds_1,DFDF_test_ds_2]
```

```
name_models=["FF_Tr1",'FF_Tr1_DFDC_Tr1','FF_Tr1_DFDC_Tr1_FF_Tr2','FF_Tr1_DFDC_Tr1_FF_Tr2
name_ds=['FF_test_ds_1','FF_test_ds_2','DFDF_test_ds_1','DFDF_test_ds_2']
model_results=[]
for i in range(4):
    print("For Model : ", name_models[i])
    for j in range(4):
        result=test_models[i].evaluate(test_ds[j])
        print(f"For {name_ds[j]} dataset the accuracy is : {result[1]}")
        model_results.append({"Models": name_models[i], "Test Dataset":name_ds[j], "Accu
```

In [2]:

```
import pandas as pd
res_df=pd.DataFrame(model_results)
print(tabulate(res_df,headers='keys',tablefmt='pretty'))
```

```
+----+---------------------------------+----------------+--------------------+
|    |             Models              |  Test Dataset  |      Accuracy      |
+----+---------------------------------+----------------+--------------------+
| 0  |             FF_Tr1              |  FF_test_ds_1  | 0.8193374276161194 |
| 1  |             FF_Tr1              |  FF_test_ds_2  | 0.8131273984909058 |
| 2  |             FF_Tr1              | DFDF_test_ds_1 | 0.5499961972236633 |
| 3  |             FF_Tr1              | DFDF_test_ds_2 | 0.567733645439148  |
| 4  |         FF_Tr1_DFDC_Tr1         |  FF_test_ds_1  | 0.6151772141456604 |
| 5  |         FF_Tr1_DFDC_Tr1         |  FF_test_ds_2  | 0.546332061290741  |
| 6  |         FF_Tr1_DFDC_Tr1         | DFDF_test_ds_1 | 0.6963948011398315 |
| 7  |         FF_Tr1_DFDC_Tr1         | DFDF_test_ds_2 | 0.5775928497314453 |
| 8  |     FF_Tr1_DFDC_Tr1_FF_Tr2      |  FF_test_ds_1  | 0.8424499034881592 |
| 9  |     FF_Tr1_DFDC_Tr1_FF_Tr2      |  FF_test_ds_2  | 0.7864865064620972 |
| 10 |     FF_Tr1_DFDC_Tr1_FF_Tr2      | DFDF_test_ds_1 | 0.5831003189086914 |
| 11 |     FF_Tr1_DFDC_Tr1_FF_Tr2      | DFDF_test_ds_2 | 0.5972471237182617 |
| 12 | FF_Tr1_DFDC_Tr1_FF_Tr2_DFDC_Tr2 |  FF_test_ds_1  | 0.6032357215881348 |
| 13 | FF_Tr1_DFDC_Tr1_FF_Tr2_DFDC_Tr2 |  FF_test_ds_2  | 0.6034749150276184 |
| 14 | FF_Tr1_DFDC_Tr1_FF_Tr2_DFDC_Tr2 | DFDF_test_ds_1 | 0.7213362455368042 |
| 15 | FF_Tr1_DFDC_Tr1_FF_Tr2_DFDC_Tr2 | DFDF_test_ds_2 | 0.6711267828941345 |
+----+---------------------------------+----------------+--------------------+
```

CNN+Vit 1 with Pytorch

Adding Convolutions - 1: On the patches

Instead of having a Linear Projection from the patches being fed into the Transformer encoder, we can convolve over the patches then feed them into the encoder. We can simulate this by having both kernel size and stride be equal to the size of the patches.

In [1]:
```
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader
from torchvision import transforms
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torch.backends.cudnn as cudnn
from tqdm import tqdm
from einops import rearrange, repeat
from einops.layers.torch import Rearrange
```

In [2]:
```
TRAIN = "/home/udai/Desktop/Data_FF/train_1"
VAL = "/home/udai/Desktop/Data_FF/test_1"

INPUT_HEIGHT = 256
INPUT_WIDTH = 256

BATCH_SIZE = 64
VAL_SPLIT = 0.1
EPOCH=25
```

In [3]:
```
resize = transforms.Resize(size=(INPUT_HEIGHT,INPUT_WIDTH))
hFlip = transforms.RandomHorizontalFlip(p=0.25)
vFlip = transforms.RandomVerticalFlip(p=0.25)
rotate = transforms.RandomRotation(degrees=15)

trainTransforms = transforms.Compose([resize, hFlip, vFlip, rotate,transforms.ToTensor()
valTransforms = transforms.Compose([resize, transforms.ToTensor()])

trainDataset = ImageFolder(root=TRAIN,transform=trainTransforms,)
valDataset = ImageFolder(root=VAL, transform=valTransforms)

print("Training dataset contains {} samples...".format(len(trainDataset)))
print("Validation dataset contains {} samples...".format(len(valDataset)))

trainloader = DataLoader(trainDataset,batch_size=BATCH_SIZE, shuffle=True)
testloader = DataLoader(valDataset, batch_size=BATCH_SIZE)
```

```
Training dataset contains 10440 samples...
Validation dataset contains 2596 samples...
```

In [4]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)
criterion = nn.CrossEntropyLoss()
scaler = torch.cuda.amp.GradScaler(enabled=False)
```

```
cuda
```

In [5]:
```
def train(net, epoch):
    print('\nEpoch: %d' % epoch)
```

```
    optimizer = optim.Adam(net.parameters(), lr=1e-4)
    net.train()
    train_loss = 0
    correct = 0
    total = 0
    print("Start Training")
    for batch_idx, (inputs, targets) in enumerate(tqdm(trainloader)):
        inputs, targets = inputs.to(device), targets.to(device)
        # Train with amp
        with torch.cuda.amp.autocast(enabled=False):
            outputs = net(inputs)
            loss = criterion(outputs, targets)
        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()
        optimizer.zero_grad()

        train_loss += loss.item()
        _, predicted = outputs.max(1)
        total += targets.size(0)
        correct += predicted.eq(targets).sum().item()

#         if batch_idx == len(trainloader) - 1:
#               print('train', batch_idx, len(trainloader), 'Loss: %.3f | Acc: %.3f%% (%
#                 % (train_loss/(batch_idx+1), 100.*correct/total, correct, total))

    return round(train_loss/(batch_idx+1),2),round(100.*correct/total,2)


def test(net, epoch):
    global best_acc
    net.eval()
    test_loss = 0
    correct = 0
    total = 0
    with torch.no_grad():
        for batch_idx, (inputs, targets) in enumerate(tqdm(testloader)):
            inputs, targets = inputs.to(device), targets.to(device)
            outputs = net(inputs)
            loss = criterion(outputs, targets)

            test_loss += loss.item()
            _, predicted = outputs.max(1)
            total += targets.size(0)
            correct += predicted.eq(targets).sum().item()

#             if batch_idx == len(testloader) - 1:
#                   print('test', batch_idx, len(testloader), 'Loss: %.3f | Acc: %.3f%%
#                     % (test_loss/(batch_idx+1), 100.*correct/total, correct, total))
    acc = round(100.*correct/total,2)

    return round(test_loss,2), acc

def pair(t):
    return t if isinstance(t, tuple) else (t, t)
```

In [6]:

```
class PreNorm(nn.Module):
    def __init__(self, dim, fn):
        super().__init__()
        self.norm = nn.LayerNorm(dim)
        self.fn = fn

    def forward(self, x, **kwargs):
        return self.fn(self.norm(x), **kwargs)
```

```
class FeedForward(nn.Module):
    def __init__(self, dim, hidden_dim, dropout = 0.):
        super().__init__()
        self.net = nn.Sequential(
          nn.Linear(dim, hidden_dim),
          nn.GELU(),
          nn.Dropout(dropout),
          nn.Linear(hidden_dim, dim),
          nn.Dropout(dropout)
        )

    def forward(self, x):
        return self.net(x)
```

In [7]:
```
class Attention(nn.Module):
    def __init__(self, dim, heads = 8, dim_head = 64, dropout = 0.):
        super().__init__()
        inner_dim = dim_head * heads
        project_out = not (heads == 1 and dim_head == dim)

        self.heads = heads
        self.scale = dim_head ** -0.5

        self.attend = nn.Softmax(dim = -1)
        self.to_qkv = nn.Linear(dim, inner_dim * 3, bias = False)

        self.to_out = nn.Sequential(
          nn.Linear(inner_dim, dim),
          nn.Dropout(dropout)
          ) if project_out else nn.Identity()
    def forward(self, x):
        qkv = self.to_qkv(x).chunk(3, dim = -1)
        q, k, v = map(lambda t: rearrange(t, 'b n (h d) -> b h n d', h = self.heads), qk

        dots = torch.matmul(q, k.transpose(-1, -2)) * self.scale

        attn = self.attend(dots)

        out = torch.matmul(attn, v)
        out = rearrange(out, 'b h n d -> b n (h d)')
        return self.to_out(out)
```

In [8]:
```
class Transformer(nn.Module):
    def __init__(self, dim, depth, heads, dim_head, mlp_dim, dropout = 0.):
        super().__init__()
        self.layers = nn.ModuleList([])
        for _ in range(depth):
            self.layers.append(nn.ModuleList([
                PreNorm(dim, Attention(dim, heads = heads, dim_head = dim_head, dropout
                PreNorm(dim, FeedForward(dim, mlp_dim, dropout = dropout))
                ]))
    def forward(self, x):
        for attn, ff in self.layers:
            x = attn(x) + x
            x = ff(x) + x
            return x
```

In [9]:
```
class ViTConv1(nn.Module):
    def __init__(self, *, image_size, patch_size, num_classes, dim, depth, heads, mlp_di
        super().__init__()
        image_height, image_width = pair(image_size)
        patch_height, patch_width = pair(patch_size)
```

```
        assert image_height % patch_height == 0 and image_width % patch_width == 0, 'Ima

        num_patches = (image_height // patch_height) * (image_width // patch_width) # 64
        patch_dim = channels * patch_height * patch_width # 48
        assert pool in {'cls', 'mean'}, 'pool type must be either cls (cls token) or mea

        self.to_patch_embedding = nn.Sequential(
          nn.Conv2d(3, 3, kernel_size=patch_height, stride=patch_height), # -> 512, 3, 8
          Rearrange('b c h w -> b (h w) c'), # -> 512 64 3
          nn.Linear(3, dim),
          )

        self.pos_embedding = nn.Parameter(torch.randn(1, num_patches + 1, dim))
        self.cls_token = nn.Parameter(torch.randn(1, 1, dim))
        self.dropout = nn.Dropout(emb_dropout)

        self.transformer = Transformer(dim, depth, heads, dim_head, mlp_dim, dropout)

        self.pool = pool
        self.to_latent = nn.Identity()

        self.mlp_head = nn.Sequential(
            nn.LayerNorm(dim),
            nn.Linear(dim, num_classes)
        )

    def forward(self, img):
        x = self.to_patch_embedding(img)
        b, n, _ = x.shape
        cls_tokens = repeat(self.cls_token, '() n d -> b n d', b = b)
        x = torch.cat((cls_tokens, x), dim=1)
        x += self.pos_embedding[:, :(n + 1)]
        x = self.dropout(x)

        x = self.transformer(x)

        x = x.mean(dim = 1) if self.pool == 'mean' else x[:, 0]

        x = self.to_latent(x)
        return self.mlp_head(x)
```

In [13]:
```
net2 = ViTConv1(
    image_size = INPUT_HEIGHT,
    patch_size = 16,
    num_classes = 2,
    dim = 512,
    depth = 32,
    heads = 16,
    mlp_dim = 512,
    dropout = 0.1,
    emb_dropout = 0.1
)
net2 = torch.nn.DataParallel(net2)
```

In [14]:
```
from torchsummary import summary
summary(net2,(3,256,256))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 3, 16, 16]           2,307
         Rearrange-2               [-1, 256, 3]               0
            Linear-3             [-1, 256, 512]           2,048
           Dropout-4             [-1, 257, 512]               0
         LayerNorm-5             [-1, 257, 512]           1,024
            Linear-6            [-1, 257, 3072]       1,572,864
           Softmax-7         [-1, 16, 257, 257]               0
            Linear-8             [-1, 257, 512]         524,800
           Dropout-9             [-1, 257, 512]               0
         Attention-10             [-1, 257, 512]               0
           PreNorm-11             [-1, 257, 512]               0
         LayerNorm-12             [-1, 257, 512]           1,024
            Linear-13             [-1, 257, 512]         262,656
              GELU-14             [-1, 257, 512]               0
           Dropout-15             [-1, 257, 512]               0
            Linear-16             [-1, 257, 512]         262,656
           Dropout-17             [-1, 257, 512]               0
       FeedForward-18             [-1, 257, 512]               0
           PreNorm-19             [-1, 257, 512]               0
       Transformer-20             [-1, 257, 512]               0
          Identity-21                  [-1, 512]               0
         LayerNorm-22                  [-1, 512]           1,024
            Linear-23                    [-1, 2]           1,026
          ViTConv1-24                    [-1, 2]               0
================================================================
Total params: 2,631,429
Trainable params: 2,631,429
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.75
Forward/backward pass size (MB): 30.16
Params size (MB): 10.04
Estimated Total Size (MB): 40.95
----------------------------------------------------------------
```

In [15]:

```python
import datetime
import os
time=datetime.datetime.now()
output_folder="CNN+VIT_Output_at_{}-{}-{}_H{}M{}".format(time.day,time.month,time.year,t

os.mkdir(output_folder)
cwd=os.getcwd()


for i in range(EPOCH):
    t_loss,t_acc=train(net2, i)
    v_loss,v_acc=test(net2, i)
    print(f"Epoch : {i} Train_loss : {t_loss} Train_acc : {t_acc} Test_loss : {v_loss} T
    with open(file=os.path.join(cwd,output_folder,"model_logs.txt"), mode="a") as log:
        log.write(f"Epoch : {i} Train_loss : {t_loss} Train_acc : {t_acc} Test_loss : {v
    torch.save(net2,os.path.join(cwd,output_folder,f"model_at_epoch_{i}.model"))
```

```
Epoch: 0
Start Training
100%|██████████| 164/164 [02:23<00:00,  1.14it/s]
100%|██████████| 41/41 [00:17<00:00,  2.30it/s]
Epoch : 0 Train_loss : 0.71 Train_acc : 50.07 Test_loss : 28.46 Test_acc 50.77

Epoch: 1
Start Training
```

```
100%|██████████| 164/164 [01:18<00:00,  2.09it/s]
100%|██████████| 41/41 [00:16<00:00,  2.53it/s]
Epoch : 1 Train_loss : 0.7 Train_acc : 49.93 Test_loss : 28.54 Test_acc 49.38

Epoch: 2
Start Training
100%|██████████| 164/164 [01:16<00:00,  2.15it/s]
100%|██████████| 41/41 [00:16<00:00,  2.54it/s]
Epoch : 2 Train_loss : 0.7 Train_acc : 50.29 Test_loss : 28.73 Test_acc 49.23

Epoch: 3
Start Training
100%|██████████| 164/164 [01:15<00:00,  2.17it/s]
100%|██████████| 41/41 [00:16<00:00,  2.54it/s]
Epoch : 3 Train_loss : 0.7 Train_acc : 50.1 Test_loss : 28.62 Test_acc 46.46

Epoch: 4
Start Training
100%|██████████| 164/164 [01:16<00:00,  2.16it/s]
100%|██████████| 41/41 [00:15<00:00,  2.65it/s]
Epoch : 4 Train_loss : 0.7 Train_acc : 50.8 Test_loss : 29.17 Test_acc 50.77

Epoch: 5
Start Training
100%|██████████| 164/164 [01:15<00:00,  2.17it/s]
100%|██████████| 41/41 [00:15<00:00,  2.69it/s]
Epoch : 5 Train_loss : 0.7 Train_acc : 51.43 Test_loss : 28.75 Test_acc 48.07

Epoch: 6
Start Training
100%|██████████| 164/164 [01:17<00:00,  2.12it/s]
100%|██████████| 41/41 [00:16<00:00,  2.49it/s]
Epoch : 6 Train_loss : 0.7 Train_acc : 51.88 Test_loss : 28.74 Test_acc 46.11

Epoch: 7
Start Training
100%|██████████| 164/164 [01:16<00:00,  2.15it/s]
100%|██████████| 41/41 [00:15<00:00,  2.60it/s]
Epoch : 7 Train_loss : 0.69 Train_acc : 52.75 Test_loss : 29.27 Test_acc 48.04

Epoch: 8
Start Training
100%|██████████| 164/164 [01:16<00:00,  2.14it/s]
100%|██████████| 41/41 [00:16<00:00,  2.52it/s]
Epoch : 8 Train_loss : 0.69 Train_acc : 52.89 Test_loss : 28.95 Test_acc 51.85

Epoch: 9
Start Training
100%|██████████| 164/164 [01:15<00:00,  2.17it/s]
100%|██████████| 41/41 [00:15<00:00,  2.61it/s]
Epoch : 9 Train_loss : 0.69 Train_acc : 53.05 Test_loss : 29.31 Test_acc 45.42

Epoch: 10
Start Training
100%|██████████| 164/164 [01:16<00:00,  2.14it/s]
100%|██████████| 41/41 [00:15<00:00,  2.62it/s]
Epoch : 10 Train_loss : 0.69 Train_acc : 53.45 Test_loss : 29.34 Test_acc 49.35

Epoch: 11
Start Training
```

```
100%|██████████| 164/164 [01:16<00:00,  2.15it/s]
100%|██████████| 41/41 [00:15<00:00,  2.62it/s]
Epoch : 11 Train_loss : 0.69 Train_acc : 53.26 Test_loss : 28.95 Test_acc 43.99

Epoch: 12
Start Training
100%|██████████| 164/164 [01:16<00:00,  2.15it/s]
100%|██████████| 41/41 [00:15<00:00,  2.61it/s]
Epoch : 12 Train_loss : 0.69 Train_acc : 53.08 Test_loss : 28.73 Test_acc 51.12

Epoch: 13
Start Training
100%|██████████| 164/164 [01:15<00:00,  2.18it/s]
100%|██████████| 41/41 [00:16<00:00,  2.54it/s]
Epoch : 13 Train_loss : 0.69 Train_acc : 53.72 Test_loss : 29.58 Test_acc 49.46

Epoch: 14
Start Training
100%|██████████| 164/164 [01:17<00:00,  2.12it/s]
100%|██████████| 41/41 [00:15<00:00,  2.59it/s]
Epoch : 14 Train_loss : 0.69 Train_acc : 54.18 Test_loss : 30.36 Test_acc 46.92

Epoch: 15
Start Training
100%|██████████| 164/164 [01:15<00:00,  2.17it/s]
100%|██████████| 41/41 [00:15<00:00,  2.60it/s]
Epoch : 15 Train_loss : 0.68 Train_acc : 54.72 Test_loss : 29.8 Test_acc 43.72

Epoch: 16
Start Training
100%|██████████| 164/164 [01:16<00:00,  2.15it/s]
100%|██████████| 41/41 [00:15<00:00,  2.59it/s]
Epoch : 16 Train_loss : 0.68 Train_acc : 54.93 Test_loss : 30.28 Test_acc 45.11

Epoch: 17
Start Training
100%|██████████| 164/164 [01:16<00:00,  2.14it/s]
100%|██████████| 41/41 [00:15<00:00,  2.57it/s]
Epoch : 17 Train_loss : 0.68 Train_acc : 56.09 Test_loss : 31.46 Test_acc 39.06

Epoch: 18
Start Training
100%|██████████| 164/164 [01:17<00:00,  2.11it/s]
100%|██████████| 41/41 [00:16<00:00,  2.56it/s]
Epoch : 18 Train_loss : 0.68 Train_acc : 55.62 Test_loss : 30.89 Test_acc 42.87

Epoch: 19
Start Training
100%|██████████| 164/164 [01:16<00:00,  2.14it/s]
100%|██████████| 41/41 [00:16<00:00,  2.51it/s]
Epoch : 19 Train_loss : 0.67 Train_acc : 57.07 Test_loss : 31.31 Test_acc 45.22

Epoch: 20
Start Training
100%|██████████| 164/164 [01:16<00:00,  2.15it/s]
100%|██████████| 41/41 [00:15<00:00,  2.66it/s]
Epoch : 20 Train_loss : 0.67 Train_acc : 57.1 Test_loss : 34.1 Test_acc 44.61

Epoch: 21
Start Training
```

```
100%|██████████| 164/164 [01:15<00:00,  2.17it/s]
100%|██████████| 41/41 [00:15<00:00,  2.60it/s]
Epoch : 21 Train_loss : 0.67 Train_acc : 57.19 Test_loss : 32.89 Test_acc 42.68

Epoch: 22
Start Training
100%|██████████| 164/164 [01:15<00:00,  2.18it/s]
100%|██████████| 41/41 [00:15<00:00,  2.66it/s]
Epoch : 22 Train_loss : 0.66 Train_acc : 57.38 Test_loss : 34.8 Test_acc 45.3

Epoch: 23
Start Training
100%|██████████| 164/164 [01:17<00:00,  2.13it/s]
100%|██████████| 41/41 [00:15<00:00,  2.69it/s]
Epoch : 23 Train_loss : 0.66 Train_acc : 57.45 Test_loss : 36.64 Test_acc 42.8

Epoch: 24
Start Training
100%|██████████| 164/164 [01:14<00:00,  2.19it/s]
100%|██████████| 41/41 [00:15<00:00,  2.66it/s]
Epoch : 24 Train_loss : 0.66 Train_acc : 57.65 Test_loss : 37.13 Test_acc 40.33
```

In [ ]:

Code For Testing Videos

In [2]:
```
def video_detector(filename, keras_model,haar_cascade):
    import cv2
    import os
    import numpy as np
    import keras
    from tqdm import tqdm


    tempfolder=f"temp{np.random.random()}"
    if not os.path.exists('./'+tempfolder) :
        os.mkdir('./'+tempfolder)
        os.mkdir('./'+tempfolder+'/real')
        os.mkdir('./'+tempfolder+'/fake')


    # Open video file
    vidcap = cv2.VideoCapture(filename)
    no_of_frames = int(vidcap.get(cv2.CAP_PROP_FRAME_COUNT))
    pbar = tqdm(total=no_of_frames)

    count = 0
    success, image = vidcap.read()
    while success:
       # if in_frame <= count < out_frame:
        gray_img = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        faces_rect = haar_cascade.detectMultiScale(gray_img, scaleFactor=1.1, minNeighbors=40)
        #print(faces_rect)
        for (x, y, w, h) in faces_rect:
           x,y,w,h=x,y,w,h
           break

        # Save the frame with detected faces
        try:
            face=image[y-10:y + 289, x:x + 299]
            prediction=keras_model.predict(np.array([cv2.resize(face, (299,299))]),verbose=0)
            pred_label=np.argmax(prediction[0])
            output_filename = os.path.join('./',tempfolder,'real',f"frame_{count}_{pred_label}.jpg")
            cv2.imwrite(output_filename,face )
        except:
            #print("no face found")
            pass
        pbar.update(1)
        success, image = vidcap.read()
        count += 1
        if count>=100:
            break


    vidcap.release()
    pbar.close()
    try:
        test_ds=keras.utils.image_dataset_from_directory(
            './'+tempfolder,
            labels="inferred",
            label_mode="categorical",
            batch_size=32,
            image_size=(299, 299),
            shuffle=False,
            )


        return (keras_model.evaluate(test_ds)[1])
    except:

        return np.nan
```

In [ ]:
```
import os
import keras
import cv2
import pandas as pd

# filename=r"C:\Users\roysu\Desktop\Project\testing1\ahbweevwpv.mp4"
haar_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
keras_model=keras.src.saving.load_model(r"C:\Users\roysu\Desktop\Project\final models\14th_May_xception_model_on_FF++DFDC and FF++Tr2_at_2.keras")

input_folder=r"C:\Users\roysu\Desktop\Project\testing"
df=pd.read_json(r"C:\Users\roysu\Desktop\Project\testing\metadata.json")

out_dict={'video_name' : [],
          'label' : [],
          'real_prob' : [],
          #'total_faces_detected' : []
          }
video_list = df.keys()
for video in (video_list[:50]):
    filename=os.path.join(input_folder,video)
    acc=(video_detector(filename, keras_model, haar_cascade))
    print(acc)
    lab=df[video]['label']
```

```
    out_dict['video_name'].append(video)
    out_dict['label'].append(lab)
    out_dict['real_prob'].append(acc)


out_df=pd.DataFrame(out_dict)

out_filename='./Output_evaluate_14_may_DFDC_2.csv'
out_df.to_csv(out_filename)

ddff=pd.read_csv(out_filename)

ddff=ddff.dropna()

ddff['pred_label']=ddff['real_prob'].apply(lambda x : 'REAL' if x>0.8 else 'FAKE')

from sklearn.metrics import classification_report
print(classification_report(ddff['label'],ddff['pred_label']))
```

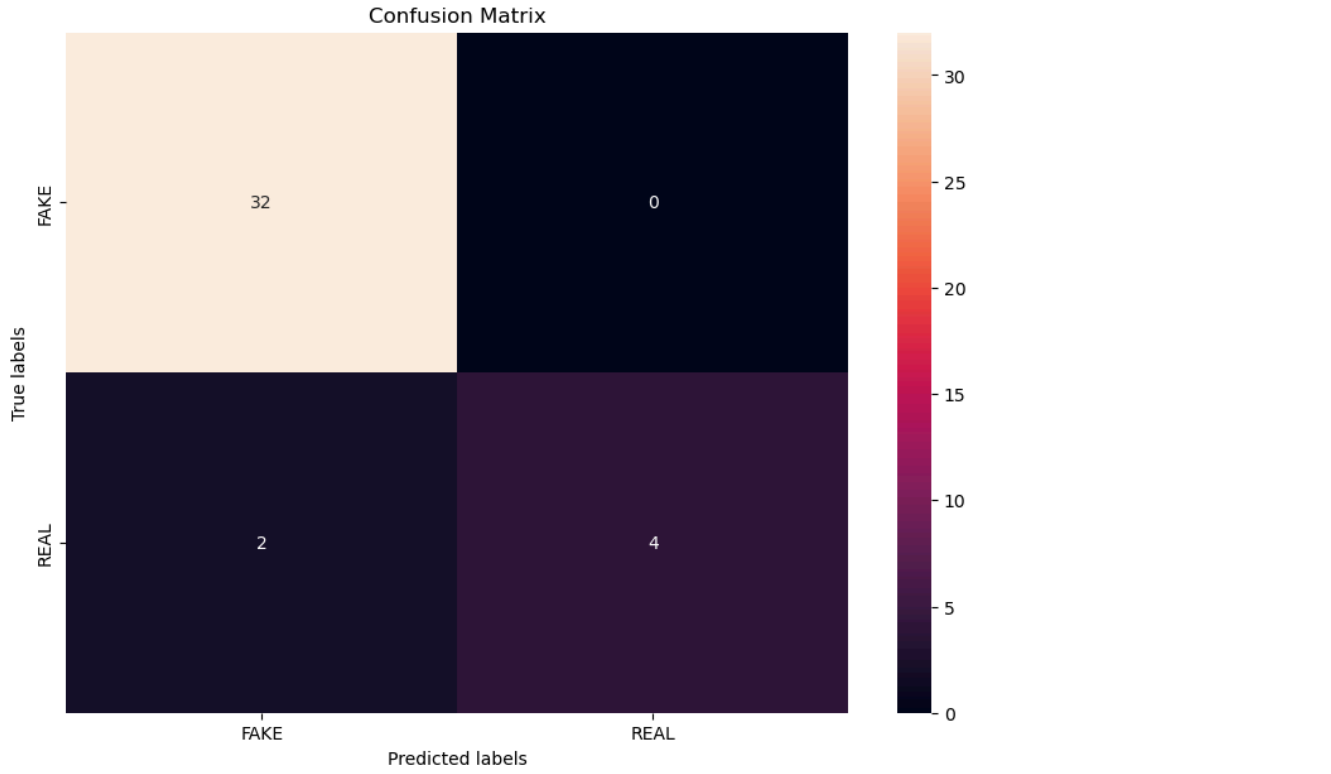
In [12]:

```
ddff['pred_label']=ddff['real_prob'].apply(lambda x : 'REAL' if x>0.8 else 'FAKE')
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
print(classification_report(ddff['label'],ddff['pred_label']))
plt.figure(figsize=(10, 7))
sns.heatmap(confusion_matrix(ddff['label'],ddff['pred_label']), annot=True, fmt='g',xticklabels=['FAKE','REAL'],yticklabels=['FAKE','REAL'])
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```

```
              precision    recall  f1-score   support

        FAKE       0.94      1.00      0.97        32
        REAL       1.00      0.67      0.80         6

    accuracy                           0.95        38
   macro avg       0.97      0.83      0.88        38
weighted avg       0.95      0.95      0.94        38
```



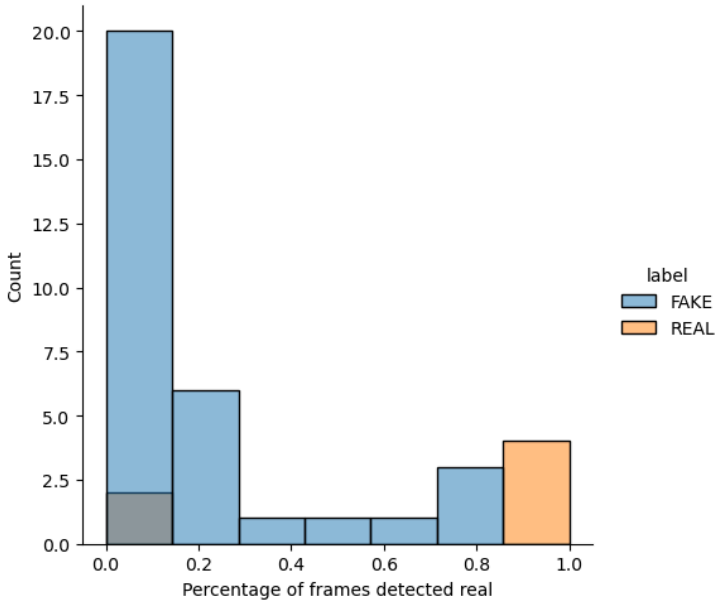
In [14]:

```
sns.displot(x=ddff['real_prob'],hue=ddff['label'])
plt.xlabel("Percentage of frames detected real")
```

```
C:\Users\roysu\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

Out[14]:

```
Text(0.5, 9.444444444444438, 'Percentage of frames detected real')
```


20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Count
0.0
0.2
0.4
0.6
0.8
1.0
Percentage of frames detected real
label
FAKE
REAL

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow import data as tf_data
import keras
from keras import models ,layers
from warnings import filterwarnings
from keras.models import load_model
filterwarnings(action='ignore')


image_size = (299, 299)
batch_size = 32

train_ds , unused_val = keras.utils.image_dataset_from_directory('../Desktop/Data/train_1',labels="inferred",label_mode="categorical",seed=1337,
                                                                 subset='both',image_size=image_size,batch_size=batch_size,validation_split=0.01)
val_ds = keras.utils.image_dataset_from_directory('../Desktop/Data/test_1',labels="inferred",label_mode="categorical",seed=1337,
                                                  image_size=image_size,batch_size=batch_size)
test_ds = keras.utils.image_dataset_from_directory('../Desktop/Data/test_2',labels="inferred",label_mode="categorical",seed=1337,
                                                   image_size=image_size,batch_size=1)
# Prefetching samples in GPU memory helps maximize GPU utilization.
train_ds = train_ds.prefetch(tf_data.AUTOTUNE)
val_ds = val_ds.prefetch(tf_data.AUTOTUNE)
test_ds = test_ds.prefetch(tf_data.AUTOTUNE)

Retrain_Xception=load_model('./7th_May_xception_model_on_FF++Tr1_at_5.keras')
epochs = 25
callbacks = [keras.callbacks.ModelCheckpoint('./8th_May_xception_model_on_FF++Tr1_at_{epoch}.keras',monitor="val_loss",mode="min",save_freq="epoch"),
           keras.callbacks.EarlyStopping(monitor="val_loss",patience=5,mode="min"),
           keras.callbacks.CSVLogger('8th_May_xception_model_on_FF++Tr1_logs.csv',append=False)]

history=Retrain_Xception.fit(train_ds,epochs=epochs,callbacks=callbacks,validation_data=val_ds)
```

```
Found 60580 files belonging to 2 classes.
Using 59975 files for training.
Using 605 files for validation.
Found 13231 files belonging to 2 classes.
Found 15620 files belonging to 2 classes.
Epoch 1/25
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 0s 12s/step - acc: 0.7420 - loss: 0.5457 2024-05-08 17:28:47.576654: W tensorflow/core/kernels/da
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 23288s 12s/step - acc: 0.7420 - loss: 0.5456 - val_acc: 0.6644 - val_loss: 0.9415
Epoch 2/25
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 23076s 12s/step - acc: 0.9369 - loss: 0.1683 - val_acc: 0.6964 - val_loss: 1.0120
Epoch 3/25
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 0s 12s/step - acc: 0.9670 - loss: 0.0947 2024-05-09 06:12:35.814654: W tensorflow/core/kernels/da
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 22657s 12s/step - acc: 0.9670 - loss: 0.0947 - val_acc: 0.6872 - val_loss: 1.4536
Epoch 4/25
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 22735s 12s/step - acc: 0.9773 - loss: 0.0658 - val_acc: 0.6482 - val_loss: 1.9159
Epoch 5/25
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 22755s 12s/step - acc: 0.9820 - loss: 0.0547 - val_acc: 0.6700 - val_loss: 1.7911
Epoch 6/25
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 23100s 12s/step - acc: 0.9862 - loss: 0.0413 - val_acc: 0.6848 - val_loss: 1.6929
```

```
Retrain_Xception.evaluate(test_ds)
```

```
2590/2590 ━━━━━━━━━━━━━━━━━━━━ 217s 84ms/step - acc: 0.7248 - loss: 2.2388
[2.273427963256836, 0.7281853556632996]
```

FIGURE 10. Retraining the Models

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow import data as tf_data
import keras
from keras import models ,layers
from warnings import filterwarnings
filterwarnings(action='ignore')
import os
import cv2
import numpy as np
from tensorflow.keras.models import load_model
from tqdm import tqdm


image_size = (299, 299)
batch_size = 32


test_ds = keras.utils.image_dataset_from_directory(
    '../Desktop/Data_FF/test_2',
    labels="inferred",
    label_mode="categorical",
    seed=1337,
    image_size=image_size,
    batch_size=1,
)

xception_model=load_model("10th_May_xception_model_on_FF++DFDC and FF++Tr2_at_3.keras")


result=xception_model.evaluate(test_ds)
print("Accuracy : ",result[1])
```

```
Found 2590 files belonging to 2 classes.
2590/2590 ━━━━━━━━━━━━━━━━━━━━ 197s 76ms/step - acc: 0.7870 - loss: 0.7753
Accuracy :  0.7864865064620972
```

```
xception_model=load_model("8th_May_xception_model_on_FF++Tr1_at_2.keras")


result=xception_model.evaluate(test_ds)
print("Accuracy : ",result[1])
```

```
2590/2590 ━━━━━━━━━━━━━━━━━━━━ 204s 78ms/step - acc: 0.5409 - loss: 1.5917
Accuracy :  0.546332061290741
```

```
Start coding or generate with AI.
```

FIGURE 11. Catastrophic Forgetting