DeepFake Detection

Ana Tomash
Shorouq Alasbali
Norah Alamri

December 2, 2023

1 Introduction

1.1 Problem description

Deepfake technology has emerged as a groundbreaking project in artificial intelligence, enabling the synthesis of hyper-realistic, computer-generated audio and video content that can convincingly mimic real human expressions and voices. While deepfakes can be used for harmless purposes, such as entertainment or parody, they can also be used for malicious purposes, such as spreading misinformation or damaging someone's reputation. As deepfake technology continues to improve, it is becoming increasingly difficult to distinguish between real and fake videos.

1.2 Project Goal

This project aims to develop and apply deep-learning techniques to effectively detect deepfake videos. The model will be trained on a dataset of real and fake videos, and evaluated on its ability to distinguish between the two accurately.

2 Literature Review

Deepfake technology uses sophisticated deep-learning algorithms, particularly generative adversarial networks (GANs) and deep neural networks (DNNs), to generate convincingly realistic synthetic content, such as images and videos. These algorithms are well-known for their capacity to mimic human features, expressions, and voices with astonishing accuracy.

Moreover, there are some of the key techniques employed in the "FakeDeep" project: Deepfake Detection Algorithms: These algorithms are designed to identify deepfake content within multimedia, such as images, videos, and audio recordings. Common approaches include:

- Convolutional Neural Networks (CNNs): Utilized for image and video analysis, CNNs can learn patterns and anomalies in visual data that may indicate deepfake manipulation.[3]

- Recurrent Neural Networks (RNNs): Applied to analyze sequential data, RNNs are useful for detecting manipulated audio or video sequences.[4]
- Capsule Networks: These networks can capture hierarchical relationships in data, which can be beneficial for recognizing inconsistencies in deepfake content.[5]

3 Data Description

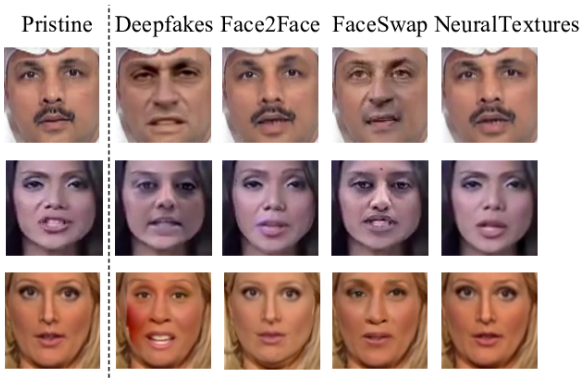


Figure 1: Sample images of pristine and manipulated images from the dataset

In this study, we utilized the FaceForensics++ dataset, specifically derived from FaceForensics-1600 videos after preprocessing. The FaceForensics++ dataset is curated for deepfake detection and encompasses genuine and manipulated facial videos. The dataset creation involved the implementation of various sophisticated techniques and tools.The manipulated sequences in the dataset were generated using four cutting-edge face manipulation techniques: Deepfakes, Faceswap, Face2Face, and Neural-Textures. These techniques were applied to produce manipulated sequences, resulting in different subsets within the dataset.

The data utilized in this study has been obtained from the publicly available platform Kaggle[7]. The dataset captured the faces in both manipulated and real videos and converted them into 300x300-sized images. The individuals responsible for creating and curating this dataset are Farhan Sharukh Hasan (Owner) and Md. Rahat Kader Khan (Editor).

Figure 2: FaceForensics++ preprocessing

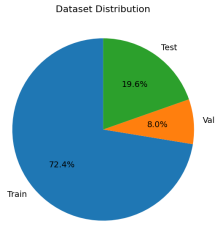
4 Dataset Split and Class Distribution

The dataset was divided into three distinct subsets for the purpose of experimentation:

1. Test Set:
   - Fake: 1541 images.
   - Real: 1562 images.
2. Training Set:
   - Fake: 5835 images.
   - Real: 5613 images.
3. Validation Set:

- Fake: 629 images.
- Real: 628 images.

The dataset used in this study exhibits a well-balanced and evenly distributed composition across its various subsets. As shown in Figure 3, we can see that the test set has 49.7% fake images and 50.3% real images, demonstrating a near-equal representation of both classes. Similarly, the training set consists of 51% fake images and 49% real images, maintaining a balanced distribution between manipulated and genuine images. The validation set further reinforces this equilibrium with 50% fake images and 50% real images. This balanced distribution of fake and real images across the test, training, and validation sets ensures that the model is exposed to a diverse and representative range of data during the training and evaluation processes. Such balanced data distribution is crucial in machine learning tasks, as it prevents biases and enables the model to generalize well to unseen data, ultimately enhancing the reliability and accuracy of the deepfake detection system developed in this research project.



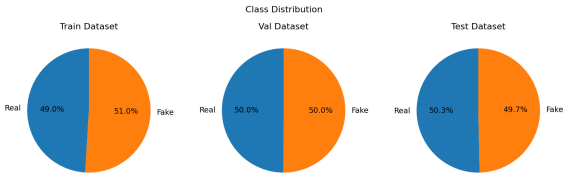


Figure 3: Class distribution of test, train, and validation sets.

5 Model Training

5.1 Utilizing CNN Models

We started with Convolutional Neural Networks (CNNs), which have proven to be highly effective in identifying deepfake content. CNNs are a class of deep learning models designed specifically for image and video analysis. Their ability to automatically learn features from images and hierarchically process them makes them a natural fit for the intricate task of deepfake detection.

Model Architecture:

- Convolutional Layers

The core of the model consists of convolutional layers, which are designed to scan and recognize local patterns within images. In our model, two sets of convolutional layers were used. The first set includes 32 filters with a size of (3, 3), followed by a second set with 64 filters of the same size. These layers apply filters to the input images, progressively learning and identifying features that are essential for the deep fake detection task.

- Max-Pooling Layers

Max-pooling layers are interwoven with the convolutional layers. After each convolutional layer, max-pooling is applied with a pooling size of (2, 2). Max-pooling reduces the spatial dimensions of feature maps, preserving the most relevant information. This step is vital for improving computational efficiency and reducing overfitting.

- Fully Connected Layers

Figure 4: CNN accuracy results for 10 epochs.

After feature extraction through convolution and pooling, the flattened output is passed through fully connected layers. We have employed two dense layers in our model. The first dense layer consists of 128 units with the Rectified Linear Unit (ReLU) activation function. The second dense layer is the output layer, with the number of units corresponding to the number of classes in our binary classification problem (in this case, "fake" and "real"). The sigmoid activation function in the output layer provides probability scores for each class, enabling the model to make predictions.

- Dropout Layer

In order to combat overfitting, a dropout layer with a rate of 0.5 was added after the first dense layer. Dropout randomly deactivates a fraction of the units during training, effectively reducing the model's reliance on any single feature and enhancing its generalization ability.

- Training Results and Hyperparameter Tuning

In our pursuit of effective deepfake detection, we selected the Adam optimizer and the adjustment of the number of training epochs.

- Optimization

The Adam optimizer is employed with a learning rate of 0.001, enhancing the training process's efficiency and convergence.

- Loss Function

The model uses binary cross-entropy as the loss function, suitable for binary classification problems such as deepfake detection.

- Evaluation Metric

The model tracks accuracy as the evaluation metric, providing insights into its ability to correctly classify deep fake and authentic content. **Model Results**

The Adam optimizer is renowned for its efficiency in optimizing deep learning models, and it played a pivotal role in enhancing our model's learning capabilities. When initially training our model with a modest 10 epochs, we achieved an accuracy rate of 61%. Realizing that extending the training duration could potentially lead to better results, we embarked on a second training iteration, this time setting the number of epochs to 30. During this extended training period, they were allowed the model to undergo more iterations and update its internal parameters, improving performance.

The outcome of your hyperparameter tuning was promising. By increasing the number of epochs from 10 to 30, our model's accuracy rose from 61% to 71.41%. This 10% improvement signifies that the model continued to learn from the dataset, refining its ability to distinguish between authentic and deepfake content.

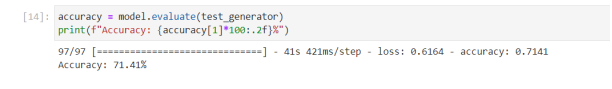


Figure 5: CNN accuracy results for 30 epochs

5.2 Utilizing Transfer Learning Methods

Transfer learning is a powerful technique in deep learning where a pre-trained neural network, trained on a large dataset for a specific task, is adapted to a different but related task. When it comes to deepfake detection, transfer learning plays a crucial role in improving the accuracy and efficiency of models.

5.2.1 MobileNetV2

MobileNetV2 is a state-of-the-art deep learning architecture designed for efficient and lightweight neural networks, particularly suited for mobile devices and edge computing applications. It was introduced by Google researchers in their paper "MobileNetV2: Inverted Residuals and Linear Bottlenecks" in 2018. MobileNetV2 builds upon the success of the original MobileNet by incorporating novel techniques to improve accuracy and efficiency.

Model Architecture:
Transfer learning using MobileNetV2 involves leveraging a pre-trained MobileNetV2 model as a feature extractor and adding custom layers on top to adapt the model for a specific task.

- Base Model (MobileNetV2 as Feature Extractor):

The MobileNetV2 model is chosen as the base model. The pre-trained weights are loaded from the 'imagenet' dataset, and the top classification layers are excluded (include_top=False).

- Custom Classification Layers:
  - Global average pooling is applied to the output of the base model.
  - A Dropout layer with a dropout rate of 0.5 is added for regularization.
  - A Dense layer with a single neuron and a sigmoid activation function is added for binary classification.
- Create the Transfer Learning Model:

Create the final transfer learning model by specifying the inputs (images) and outputs (predictions).

- Freeze Base Model Layers:

The layers of the MobileNetV2 base model are set to be non-trainable (layer.trainable = False) to retain the pre-trained weights during the initial training.

- Compile the Model:

The model is compiled using the Adam optimizer with a learning rate of 0.0001, binary crossentropy loss function, and accuracy as the evaluation metric.

- Training and Fine-Tuning:
  - The model is trained using the fit_generator method, which is deprecated but still functional in this code.
  - Training involves 50 epochs with 50 steps per epoch for training and 20 steps per epoch for validation.
- Callbacks:
  - EarlyStopping is implemented to monitor the validation loss, and training is stopped if the loss does not improve after three consecutive epochs.
  - LearningRateScheduler is defined to adjust the learning rate during training.

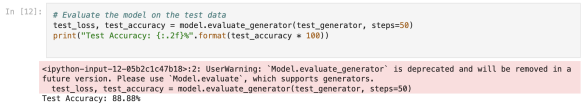


Figure 6: MobileNetV2 accuracy results for 50 epochs

Model Results

The model's performance is evaluated on the test dataset, and the results indicate that the model achieves a test accuracy of approximately 88.88%. This accuracy represents the proportion of correctly classified instances out of the total test samples.

5.2.2 MesoNet Transfer Learning Model

MesoNet is a deep learning architecture designed for facial deep face detection. It utilizes convolutional layers to capture spatiotemporal features in images, making it effective for distinguishing between real and manipulated facial images.

Model Architecture:

Base Model (Meso4 as Feature Extractor):

- Implement Meso4 as the base model for feature extraction.
- Pretrained weights are loaded to leverage knowledge from a related task.

Convolutional Layers:

- Utilize convolutional layers with Batch Normalization, MaxPooling, and LeakyReLU activations.
- Configure multiple convolutional blocks to capture hierarchical features.

Flatten and Dense Layers:

- Flatten the output to a 1D tensor.
- Apply dropout for regularization.
- Add Dense layers with LeakyReLU activations for non-linearity.

Custom Output Layer:

- Include a custom output layer with a sigmoid activation for binary classification (real or deep fake).

Transfer Learning Initialization:

- Instantiate MesoNet and load pretrained weights ('Meso4_DF').

Model Training and Results:

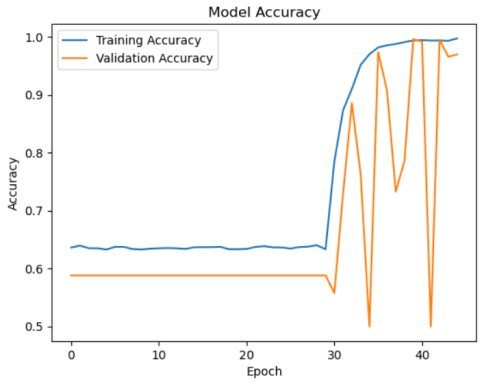


Figure 7: MesoNet Tain/Validation accuracy results.

- Train the model on the training set using the fit method with freezing the pretrained layers during the initial training and the accuracy almost 63. Then unfreezing the weights for fine-tuning and the accuracy increased to 96.

```
# Print final accuracy on the test set
final_accuracy = meso.get_accuracy(generator1[0][0], generator1[0][1])
print(f"Final Accuracy on Test Set: {final_accuracy[1]:.4f}")
```

```
Final Accuracy on Test Set: 0.9688
```

Figure 8: MesoNet accuracy results for 15 epochs.

5.2.3 VGG16

VGG16, short for Visual Geometry Group 16-layer model, is a deep convolutional neural network architecture designed for image classification. It was introduced by the Visual Geometry Group at the University of Oxford and presented at the ILSVRC (ImageNet Large Scale Visual Recognition Challenge) in 2014. VGG16 is known for its simplicity and uniform architecture.

VGG16 Architecture:

- We load the pre-trained VGG16 model with weights pre-trained on ImageNet. The top layer is excluded (include_top=False), and we specify the input shape to match our data.
- Freezing Layers:
- We freeze the pre-trained layers of the VGG16 model to retain the learned features during training of the new classification layers.
- New Model for Binary Classification:
- We create a new model using the Sequential API, consisting of the pre-trained VGG16 model, a Flatten layer, a Dense layer with ReLU activation, and a final Dense layer with a sigmoid activation for binary classification.
- Model Compilation:
- The model is compiled using the Adam optimizer with a learning rate of 0.001, binary cross-entropy loss function, and accuracy as the metric.

Model results: Training with Frozen Layers:

- When you initially train the model with the pre-trained layers frozen, the model learns to extract features from the images using the knowledge gained from the ImageNet dataset. The validation and test accuracies (around 75.34% and 72.06%, respectively) suggest that the model might be overfitting to the training data. It is not generalizing well to new data, indicating a need for further adjustment.

```
97/97 [==============================] - 835s 9s/step - loss: 1.3239 - accuracy: 0.7206
Test Accuracy: 0.7205929756164551
```

Figure 9: VGG16 Training with Frozen Layers

Unfreezing Last Layers:

- Unfreezing the last 10 layers (fine-tuning) allows the model to adjust its weights based on our specific task. The hope is that the model can capture task-specific features and improve its ability to generalize.

Lower Learning Rate:

- A lower learning rate (0.0001) is used during fine-tuning to prevent drastic changes to the pre-trained weights and to avoid overfitting.

Result after Fine-Tuning: The test accuracy after fine-tuning is approximately 49.66%, which is close to random guessing (50%). This suggests that the fine-tuning did not significantly improve the model's performance on the test set.

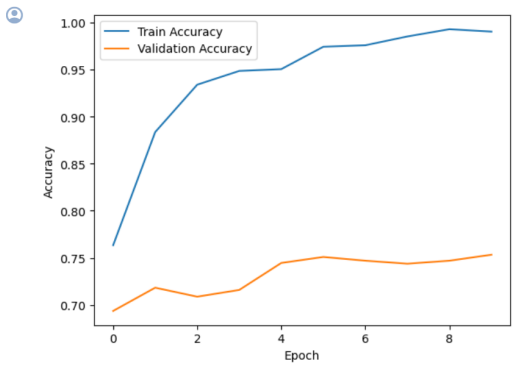


Figure 10: Plot VGG16 Training with Frozen Layers

```
# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(test_generator)
print(f'Test Accuracy: {test_acc}')
```

```
97/97 [==============================] - 832s 9s/step - loss: 0.6935 - accuracy: 0.4966
Test Accuracy: 0.4966161847114563
```

Figure 11: VGG16 Training with UnFrozen Layers

5.2.4 Xception

Xception (Extreme Inception) is a deep learning convolutional neural network (CNN) architecture that was introduced by François Chollet, the creator of the Keras deep learning library. It was proposed in the paper titled "Xception: Deep Learning with Depthwise Separable Convolutions," presented at the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) in 2017.

Xception is inspired by the Inception architecture, which is known for its parallelized and multi-scale feature extraction using various filter sizes. Xception takes this idea further by replacing the standard convolutional layers in Inception with depthwise separable convolutions. Depthwise separable convolutions consist of a depthwise convolution (which operates on each channel independently) followed by a pointwise convolution (a 1x1 convolution). This separation reduces the computational cost significantly while maintaining expressive power.

Model architecture:

Data Preprocessing:

- ImageDataGenerator is used for data augmentation and preprocessing. Augmentation includes rescaling, shearing, zooming, and horizontal flipping.

Data Generators:

- Separate generators are created for training, testing, and validation data using the specified data directories, target size, and batch size.
- The classes are specified as ['fake', 'real'], and the class_mode is set to 'binary'.

Xception Base Model:

- The Xception model is loaded with pre-trained weights from ImageNet and excludes the top classification layers.
- GlobalAveragePooling2D layer is added to reduce spatial dimensions.
- A Dense layer with one unit and a sigmoid activation function is added for binary classification.

Freezing Base Model Layers:

- The layers of the Xception base model are frozen to prevent them from being updated during training.

Model Compilation:

- The model is compiled using the Adam optimizer with a learning rate of 0.0001, binary crossentropy loss function, and accuracy as the evaluation metric.

Training:

- The model is trained with early stopping based on validation loss and a learning rate schedule defined by the lr_schedule function.
- The training process runs for 50 epochs, and the training and validation accuracy and loss are monitored.

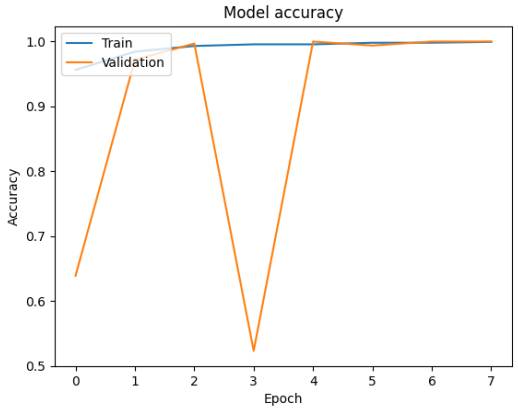


Figure 12: Plot of Xception Train/Validation Accuracy with Early Stopping

Model Result:

- The training process shows a decreasing training loss and increasing training accuracy, indicating that the model is learning from the training data.
- The validation loss and accuracy are also monitored, and early stopping is employed to prevent overfitting.
- The test accuracy achieved is very high (99.94%), suggesting that the model generalizes well to unseen data.

Figure 13: Xception Test Accuracy

6 Overall Results

In this comparative analysis of different convolutional neural network (CNN) architectures for binary image classification, the Xception model demonstrated superior performance, achieving the best test accuracy of 99.94%. This outcome highlights the efficacy of utilizing pre-trained models and transfer learning in image classification. MesoNet exhibited great performance, achieving a test accuracy of 96.88%. The MobileNetV2 architecture yielded an accuracy of 88.88%. In addition, VGG16 with frozen layers and a simpler CNN both demonstrated comparable accuracies at 72.00% and 71.41%,

Model	Test Accuracy
Xception	%99.94
MesoNet	%96.88
MobileNetV2	%88.88
VGG16 - Frozen layers	%72.00
Simple CNN	%71.41
VGG16 - Unfrozen layers	%49.66

Figure 14: Test accuracies of all models done in this project

respectively. Notably, unfreezing VGG16 layers resulted in a significant decline in performance, underscoring the nuanced nature of fine-tuning. These insights contribute valuable information to the understanding of CNN architecture selection and configuration for image classification tasks, with Xception emerging as a robust choice.

7 Future Scope

For future improvement, this project envisions several additional steps to employ:

- Applying more regress data augmentation techniques to increase training accuracy.
- Try convolutional neural networks (CNN) coupled with long short-term memory (LSTM) networks. These techniques can help us capture temporal dependencies within deep fake content, enhancing our ability to detect even more convincing forgeries.
- Utilize Face ID to localize learning parameters on face structure and patterns only reducing noise and enhancing learning accuracy.

8 Deliverables

Please, follow the link to find the comprehensive code used to create this project: GitHub final project

References

1. Rössler, Andreas, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nießner. "Face Forensics: A large-scale video dataset for forgery detection in human faces." arXiv preprint arXiv:1803.09179 (2018).

2. Hsu, Chih-Chung, Chia-Yen Lee, and Yi-Xiu Zhuang. "Learning to detect fake face images in the wild." In 2018 international symposium on computer, consumer and control (IS3C), pp. 388-391. IEEE, 2018.

3. Bonettini, Nicolo, Edoardo Daniele Cannas, Sara Mandelli, Luca Bondi, Paolo Bestagini, and Stefano Tubaro. "Video face manipulation detection through ensemble of cnns." In 2020 25th international conference on pattern recognition (ICPR), pp. 5012-5019. IEEE, 2021.

4. Güera, David, and Edward J. Delp. "Deepfake video detection using recurrent neural networks." In 2018 15th IEEE international conference on advanced video and signal based surveillance (AVSS), pp. 1-6. IEEE, 2018.

5. Nguyen, Huy H., Junichi Yamagishi, and Isao Echizen. "Capsule-forensics: Using capsule networks to detect forged images and videos." In ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 2307-2311. IEEE, 2019.

6. Rana, Md Shohel, Mohammad Nur Nobi, Beddhu Murali, and Andrew H. Sung. "Deepfake detection: A systematic literature review." IEEE access 10 (2022): 25494-25513.

7. FaceForensics-1600 videos-preprocess. (2023). Retrieved October 30, 2023, from Kaggle

8. Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).