# Deepfake detection using ResNet

Jeeva Krishnasamy

School of Computing and Information, University of Pittsburgh

INFSCI 2915 Artificial Intelligence in Digital Multimedia and Cyber Forensics

December 15, 2023

## Introduction

This report presents the implementation and evaluation of a deepfake detection system using a pre-trained InceptionResNetV2 model with fine-tuning and image augmentation. The goal is to classify images extracted from videos as either real or fake based on facial features.

## Data Preprocessing

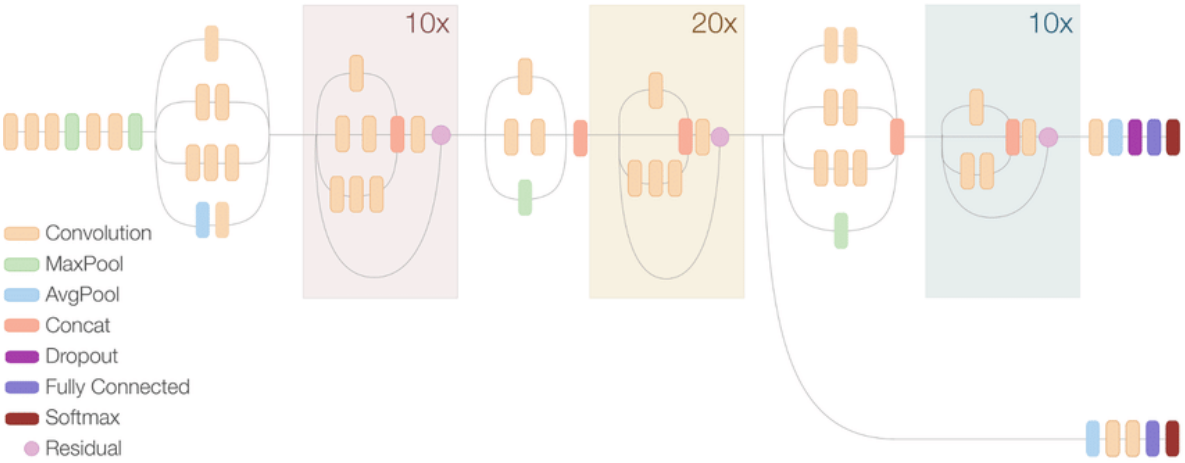
The dataset used for training and testing is divided into real and fake videos. The faces are detected using the dlib library, and the frames are sampled from the videos. Cropped face images are saved for further processing. The faces from both real and fake videos are cropped and saved for the further processing of the data.

```
# Function to save cropped faces
def save_cropped_faces(frame, count, label, vid):
    face_rects, _, _ = detector.run(frame, 0)
    for i, d in enumerate(face_rects):
        x1 = d.left()
        y1 = d.top()
        x2 = d.right()
        y2 = d.bottom()
        crop_img = frame[y1:y2, x1:x2]
        save_path = os.path.join(base_dataset_folder, f'{label}/{vid.split(".")[0]}_{count}.png')
        cv2.imwrite(save_path, cv2.resize(crop_img, (128, 128)))
        print(f'Saved image at: {save_path}')
```

## Model Architecture

The deepfake detection model is based on the InceptionResNetV2 architecture. The model is trained on the cropped face images, and the last layers are fine-tuned for the specific task. The model is compiled with a custom Adam optimiser, and binary cross-entropy is used as the loss function.

## ResNet V2 Model




```
# Define input shape and data directory
input_shape = (128, 128, 3)
data_dir = '/content/drive/MyDrive/dataset'

# List real and fake data
real_data = [f for f in os.listdir(os.path.join(data_dir, 'real')) if f.endswith('.png')]
fake_data = [f for f in os.listdir(os.path.join(data_dir, 'fake')) if f.endswith('.png')]

X = []
Y = []

# Load and preprocess images
for img in real_data:
    X.append(img_to_array(load_img(os.path.join(data_dir, 'real', img), target_size=input_shape[:2])))
    Y.append(1)

for img in fake_data:
    X.append(img_to_array(load_img(os.path.join(data_dir, 'fake', img), target_size=input_shape[:2])))
    Y.append(0)

Y_val_org = Y

# Normalization
X = np.array(X) / 255.0
Y = to_categorical(Y, 2)

# Train-Test split
X_train, X_val, Y_train, Y_val = train_test_split(X, Y, test_size=0.2, random_state=5)

# Model definition
googleNet_model = InceptionResNetV2(include_top=False, weights='imagenet', input_shape=input_shape)
googleNet_model.trainable = True

model = Sequential()
model.add(googleNet_model)
model.add(GlobalAveragePooling2D())
model.add(Dense(units=2, activation='softmax'))

# Use the Adam optimizer with a custom learning rate
custom_optimizer = optimizers.Adam(learning_rate=1e-5, beta_1=0.9, beta_2=0.999, epsilon=1e-07, amsgrad=False)
model.compile(loss='binary_crossentropy', optimizer=custom_optimizer, metrics=['accuracy'])

# model.summary()
```

The code segment serves to set up a deepfake detection model using transfer learning with InceptionResNetV2. The initial steps involve preparing the dataset, where images from the 'real' and 'fake' subdirectories are loaded and preprocessed. Pixel values are normalized, and labels are encoded using one-hot encoding. The dataset is then divided into training and validation sets.

Moving on to the model architecture, InceptionResNetV2 is employed as a feature extractor, leveraging pre-trained weights from ImageNet. The model structure includes a Global Average Pooling layer for spatial reduction, followed by a Dense layer for binary classification, distinguishing between real and fake images.

To optimize the model, the Adam optimizer is utilized with a specific learning rate and hyperparameters. The model is compiled with binary cross-entropy loss and the accuracy metric, making it ready for training on the provided dataset. This code establishes a coherent pipeline encompassing data preparation, model architecture definition, and compilation, laying the foundation for subsequent training.

**Training**

The model is trained for 20 epochs with a batch size of 32. Early stopping and learning rate reduction callbacks are implemented to prevent overfitting and improve convergence. Training accuracy, validation accuracy, training loss, and validation loss are monitored throughout the training process.

```
# Training
early_stopping = EarlyStopping(
    monitor='val_loss',
    min_delta=0,
    patience=5,
    verbose=0,
    mode='auto'
)

reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.2,
    patience=2,
    min_lr=1e-6
)
```

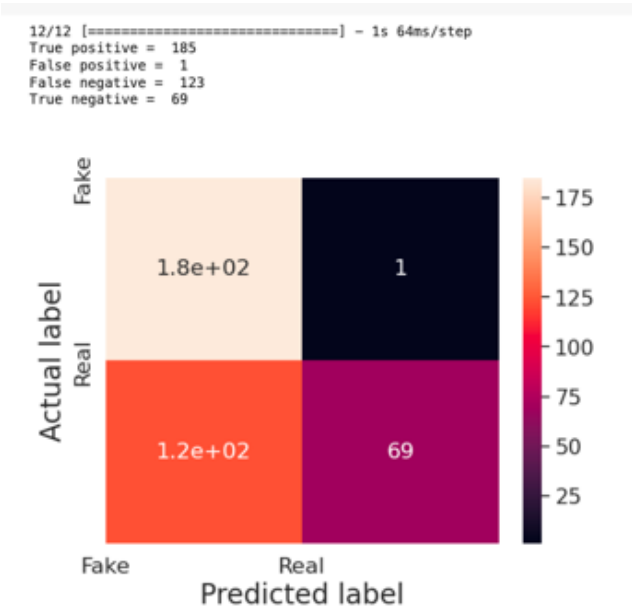
### Model Evaluation

The trained model is evaluated on a separate set of real and fake images. The confusion matrix is generated to analyze the model's performance. True positives, false positives, false negatives, and true negatives are reported, and a heatmap visualization of the confusion matrix is provided.

```
input_shape = (128, 128, 3)
pr_data = []
detector = dlib.get_frontal_face_detector()
cap = cv2.VideoCapture('/content/drive/MyDrive/Celeb-DF-v2/Celeb-synthesis/id0_id4_0002.mp4')
frameRate = cap.get(5)
while cap.isOpened():
    frameId = cap.get(1)
    ret, frame = cap.read()
    if ret != True:
        break
    if frameId % ((int(frameRate)+1)*1) == 0:
        face_rects, scores, idx = detector.run(frame, 0)
        for i, d in enumerate(face_rects):
            x1 = d.left()
            y1 = d.top()
            x2 = d.right()
            y2 = d.bottom()
            crop_img = frame[y1:y2, x1:x2]
            data = img_to_array(cv2.resize(crop_img, (128, 128))).flatten() / 255.0
            data = data.reshape(-1, 128, 128, 3)
            print(model.predict(data))
```

## Results

The training and validation accuracy and loss plots show the model's performance over the epochs. The confusion matrix provides insight into the model's ability to correctly classify real and fake images.


## Inference on New Videos

The saved model is loaded, and inference is performed on frames extracted from new videos. The predicted probabilities of being real or fake are output for each frame.

```
input_shape = (128, 128, 3)
pr_data = []
detector = dlib.get_frontal_face_detector()
cap = cv2.VideoCapture('/content/drive/MyDrive/Celeb-DF-v2/YouTube-real/00001.mp4')
frameRate = cap.get(5)
while cap.isOpened():
    frameId = cap.get(1)
    ret, frame = cap.read()
    if ret != True:
        break
    if frameId % ((int(frameRate)+1)*1) == 0:
        face_rects, scores, idx = detector.run(frame, 0)
        for i, d in enumerate(face_rects):
            x1 = d.left()
            y1 = d.top()
            x2 = d.right()
            y2 = d.bottom()
            crop_img = frame[y1:y2, x1:x2]
            data = img_to_array(cv2.resize(crop_img, (128, 128))).flatten() / 255.0
            data = data.reshape(-1, 128, 128, 3)
            print(model.predict(data))
```

```
1/1 [==============================] - 0s 38ms/step
[[0.44074854 0.5592515 ]]
```

The classification decision is determined by comparing the confidence levels generated by the model for the two classes: data (1) corresponds to the real class, and data (0) corresponds to the fake class. If the confidence level for the real class is higher than that for the fake class, it is inferred that the given video segment is real. Conversely, if the confidence level for the fake class is higher, the video segment is classified as fake. This process involves evaluating the model's predictions and making decisions based on the relative confidence levels for real and fake classes in each half of the video.

**Conclusion**

The deepfake detection model demonstrates promising results based on the evaluation metrics. Further improvements could involve tuning hyperparameters, exploring different architectures, or collecting a more diverse dataset. Deployment considerations and real-world testing are also crucial to assess the model's robustness and reliability in detecting deepfake content.