DEEPFAKE DETECTION USING DEEP LEARNING

Report
Submitted in partial fulfillment of the requirements of
CS F376 Design Project
By

KEANE COUTINHO
IDNO: 2021A7PS0080U

Under the supervision of
Dr.TAMIZHARASAN P S
Professor

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

DUBAI CAMPUS, DUBAI UAE

December - 2023

ACKNOWLEDGEMENTS

I would like to express my deepest sense of gratitude, first and foremost, to my Supervisor **Dr.Tamizharasan P S**, Professor, Computer Science Department, BITS Pilani, Dubai campus, United Arab Emirates, for his valuable guidance and encouragement during the course of this Project. I am extremely grateful to him for his able guidance, valuable technical inputs, and useful suggestions.

I express sincere thanks and gratitude to my Project Supervisor: **Ms.Prathiba P G**, our Director, BITS Pilani, Dubai Campus, **Prof. Dr. M.P. Srinivasan,** Project Ins/ i/c Dr. B. Vijayakumar for their motivation, encouragement, and support to pursue my Project.

I am grateful to the examiners for their valuable suggestions.

Above all, I thank the almighty for giving me the strength to carry out this work to the best of my abilities.

Name : KEANE COUTINHO
ID No. : 2021A7PS0080U

CERTIFICATE

This is to certify that the Mid Semester Project Report entitled, **Deepfake Detection using Deep Learning** and submitted by **KEANE COUTINHO ID No. 2021A7PS0080U** in partial fulfillment of the requirement of the CS F376 Design Project embodies the work done by him under my supervision.

Date: 24th December

Signature of the Supervisor

Name: Dr Tamizharasan P S
Designation: Professor (CS)

LIST OF FIGURES

LIST OF ABBREVIATIONS

1.	C3D	-	3-Dimensional Convolutional Network
2.	CNN	-	Convolutional Neural Network
3.	GAN	-	Generative Adversarial Networks
4.	CLRNet	-	Convolutional LSTM-based Residual Network
5.	RNN	-	Recurrent Neural Network

BITS Pilani, Dubai Campus

First Semester 2023-2024

Project Course Code and Course Name: CS F376 Design Project
Semester: I Semester 2023-2024
Duration: 28/08/2023 - 24/12/2023
Date of Start: 28/08/2023
Date of Submission: 24/12/2023
Title of the Report: Deepfake Detection using Deep learning

ID No. / Name of the student: **2021A7PS0080U / KEANE COUTINHO.**

Discipline of Student: B.E Computer Science

Name of the Project Supervisor: Dr. TAMIZHARASAN P S

Key Words: **Deepfake, Face-recognition, C3D, Detection, Binary Classification, Deep learning, GAN and CNN**

Project Area: **Data Mining**

Abstract: **The growth of Deepfake technology poses a huge potential danger to multimedia content authenticity, demanding the development of Deepfake detection systems. The deepfake detection systems are some of the most recent deep learning-powered applications to be discovered. Deepfakes are manipulated, high-quality, realistic videos or images that have acquired a lot of popularity recently. This research proposes a complete approach to deepfake detection that focuses on facial feature extraction and classification utilizing advanced neural network models. The approach entails a multi-step method meant to improve the accuracy of deepfake identification. Initially, the footage is pre-processed to obtain facial regions, allowing for more targeted analysis. After pre-processing, a C3D model and a CNN model are used to extract temporal and spatial features from the facial areas. The retrieved features are then fed into a binary classification algorithm that has been trained to recognize the deepfake videos**

Signature of Student **Signature of Supervisor**

Date: 24/12/2023 **Date: 24/12/2023**

TABLE OF CONTENTS

1. Introduction

In this day and age of rapid technological advancement, the rise of Deep Fakes shows the paradigm shift in the landscape of digital media manipulation. Deepfakes are artificial media that use Artificial Intelligence and Machine Learning techniques, particularly GANs, and deep neural networks are used to create and manipulate content mostly in the form of videos or images. The process of creating deep fakes is as follows:

- Data Collection: The Machine learning model is trained on large datasets of videos and images featuring a certain person mostly celebrities. The model learns the facial expressions and unique features of the person.
- Training the model: Deep Learning Algorithms like deep neural networks are used to train the model to where it learns to map and recreate the person's facial expressions.
- Face Swapping: Once the model is trained, it can be used to swap faces in existing videos or images.

Deepfakes have gained popularity due to their ability to produce compelling and false information. While they can be used for entertainment, they also bring ethical concerns, particularly when used to spread false information, or misinformation, or distort people's perspectives. Deepfakes have been used to impose celebrities onto sexual content, resulting in faked videos that take advantage of both popular figures' likeness and vulnerability. Deepfakes have been used in the political arena to disseminate manufactured speeches, modifying the perceived words and positions of significant figures.

Generative Adversarial Networks (GAN) are most commonly used for deep fake generation. The GAN architecture comprises 2 parts: an encode-decoder pair. The encoder is used by the model to train on the large set of data collection and generate fictional data. The decoder's main purpose is to distinguish between true and fake data. To generate a realistic face, the model demands a large amount of input. These fake

videos are then used to train the decoder which acts as a binary classifier that employs a Softmax function to discriminate between authentic and false films. FakeApp is very similar to GANs, it also uses An autoencoder to extract latent features from photos of human faces, whereas a decoder retrieves those same features. Another well-known method is called VGGFace which uses GAN. It simply adds two additional layers called adversarial loss and perceptual loss.

Due to the ongoing progress of deep learning algorithms and the sophistication of the tools used to create unauthorized material, detecting deep fakes presents multiple challenges. The increasing sophistication of deepfake algorithms, which use neural networks to generate accurate and convincing synthetic media, poses a major concern. As deepfake technology advances, distinguishing between modified and legitimate content gets more difficult. Furthermore, the vast amount and diversity of internet information makes developing universal detection algorithms difficult.

Deep learning algorithms, particularly neural networks, excel at detecting nuanced patterns and abnormalities in large datasets, which is critical for detecting tiny distortions and inconsistencies produced during deep fake production. In image and video analysis, CNNs and RNNs are extensively used, allowing models to detect subtle face traits, motions, and temporal dynamics indicative of manipulation.

2. Literature Survey

Rana et.al [1] discusses the different methods through which Deepfake detection can be done. Machine learning approaches, for example, DT, and RF show the decision process in the form of a tree. GANs are also used for creating Deepfakes but these deepfakes have irregularities that can be detected easily by MLPs. Machine Learning models achieve an accuracy of 98% in detecting deep fakes. Using Deep learning includes using a GAN simulator network to extract data from RGB data. Initially, Deep Learning methods were used for Deepfake detection. Two models used include Meso-4 & MesoInception-4. The paper shows that deeper CNNs perform much better, especially in

terms of extracting features spatio-temporal features, common textures, and face landmarks from the video. A network called CN (Capsule network) achieved an accuracy of 99% accuracy & it needs much fewer parameters to train. Many authors also suggested using CNNs for feature extraction. Statistical-based methods using PRNU are a distinct noise pattern in digital photographs caused by flaws in the camera's light-sensitive sensors. It is called the fingerprint of a digital photo. Each video frame has a certain PRNU sequence. By dividing into 8 equal groups & using second-order-FSTV methods we make a standard PRNU. After that, it is correlating measuring the normalized cross-correlation score.

Tariq et.al [2] Developed a CLRNet that addresses the limitation of methods like lack of generalizability as they are specific only for certain methods. CLRNet captures temporal information by taking a sequence of consecutive images from a video as input which helps with the detection of deepfakes. The dataset used is Face Forensics++. From the frame-by-frame analysis of the video, it is discovered that the inconsistencies found in the video include sudden changes in brightness and contrast on small sections of the face and the change in the size of particular facial components like as eyes, mouth, and eyebrows in frames. For pre-processing, 16 samples were taken from each actual and false video, with each containing 5 consecutive frames MTCNN was used to capture the faces from the extracted frame, and all the frames were resized to a 240 X 240 resolution. Other Augmentations like Zoom, Brightness, Rotation, and channel shift were done on the frames. Some baseline methods like Xception, ShallowNet, DenseNet with Bidirectional RNN, and Forensic Transfer are used to compare with their network. The methods were trained on the Deepfake base dataset and the performance is as follows, FT (99.35%) outperforms CLRNet (99.02%). Meanwhile, Xception (86%) and ShallowNet (56.65%) did not perform well.

Zhao et.al [3] called for a new multi-attentional deepfake detection network. The 3 main components include multiple spatial attention heads, textural feature enhancement blocks to zoom in on the subtle artifacts by using densely connected convolutional layers, and the low-level textural features and high-level semantic features directed by

attention maps combined. The attention block is a lightweight model that consists of a batch normalization layer, a 1x1 convolutional layer, and a non-linear activation layer. For the videos of the dataset, a face extractor called RetinaFace was used to detect the faces and save them with a size of 380 x 380. The Datasets used were FaceForensics++ and Celeb-DF. The model has an AUC (%) of 99.80% on the FaceForensics ++ and 67.44 % on the Celeb -DF dataset, the model performed better than many other architectures which include Two-Stream, Meso4, EfficientNet-B4, Two Branch, Capsule, Multi-task, Xception-raw, Xception-c23, Xception-c40 etc.

Roy et.al [4] Using pre-built architectures like 3D ResNet, 3D ResNeXt, and I3D for deep fake detection along with checking the effect of attention mechanisms, In this paper, two attention mechanisms were tested namely SE-block and non-local network. These techniques were all tested on the Face Forensics++ Dataset. The FaceForensics++ dataset has 1000 videos of talking subjects these videos have been manipulated by 4 manipulation schemes which include Faceswap, Deepfakes, Face2face, and Neural textures.I3D considers RGB frame sets as input. For spatio-temporal modeling, it replaces the original model's 2D convolutional layers with 3D convolutions. 3D ResNet and 3D ResNeXt are based on I3D and are an extension of 2D ResNet and 2D ResNeXt. All the videos were converted into 224 x 224 spatial dimensions 64 frames for the I3D, and 112 x 112 for 3D ResNet and 3D ResNeXt. After testing with all manipulation techniques I3D performed the best with a True Classification rate of 87.43 % but it was the most computationally intense. The AUC score of 3D ResNet is 0.82 but it managed to give an AUC score of 0.86 when a Non-local attention mechanism was applied to it, 3D ResNeXt with a Non-local network gave an AUC score of 0.91. When a single manipulation technique was considered I3D performed well. 3D ResNet with non-local block outperforming 3D ResNet and 3D ResNet with Attention. 3D ResNeXt improved its performance by introducing a Non-local block leading it to be one of the best-performing models among all video-based methods. From the paper, it is understood that 3D CNNs outperform other CNNs but incorporating attention mechanisms improves the detection accuracy.

Guera et.al [5] A temporal-aware pipeline for detecting deep fake videos have been proposed. The model uses a CNN to collect frame-level information before employing an RNN to understand if a video is a deep fake or not, the model RNN consists of a convolutional LSTM which is used for processing the frame sequence followed by a 512 fully -connected layer with a A softmax layer computes the odds of it being a Real or Deepfake after a 0.5 likelihood of dropout. The dataset includes 300 deepfake films from various video-hosting services. along with 300 videos from the HOHA dataset. For data pre-processing of the videos removing channel mean from each channel, resizing of every frame to 299 x 299, sub-sequence sampling of length N, N=20,40,80 frames. In terms of performance the model where N= 20 frames had a testing accuracy of 96.7%, N=40 frames had an accuracy of 97.1% and N=80 frames had an accuracy of 97.1%.

Chang *et.al* [6] Using an NA-VGG to detect deepfake face images. NA-VGG is an SRM filter & image augment layer added to VGG. The dataset being used is Celeb-DF. In terms of because feature extraction using an RGB channel is ineffective, the images are subjected to an SRM filter to recover the local noise features map from the RGB image. The image is the noise that is highlighted in the image and the classification can be done accordingly. Image Augmentation is also done to prevent overfitting of the model augmentation techniques including flipping, and rotating of images of the training set. The parameter settings used are re-sizing the Image size reduced to 128 * 128. The optimizer's parameters are SGD, 0.01, decay of 1e-6, momentum of 0.9, and Nesterov of True. The NAVGG model had an AUC score of 85.7%.

Chintha et.al [7] Use a combination of convolutional latent representation with bidirectional recurrent structure & entropy-based functions. The datasets used is Face Forensics++ & Celeb Df video datasets. The DLIB face detection algorithm detects the principal face across each frame in the video. Canonical face images are made by cropping to the facial boundary and resampling to 299 × 299 pixels in size. and a Linear smoothing filter applied to the base coordinates of successive frames to reduce temporal irregularities. Xceptemporal a convolutional recurrent network is used for temporal feature extraction which is passed into 2 layers of bidirectional LSTM, followed by a fully connected layer. 4 variants of XceptTemporal were used which are Xceptemporal

(CE), XcepTemporal (KL), XcepTemporal (EN), and XcepTemporal (EN 1+n). Overall the datasets XcepTemporal (KL) has 100% accuracy, and XcepTemporal (CE) has 99.1% accuracy. XcepTemporal (EN) had a 96.98% accuracy & XcepTemporal (EN 1+n) had an accuracy of 96.89%.

Mira et.al [8] Compare the results of using LSTM and Convolutional neural networks to distinguish between the fake from the real. The paper talks about the different deep learning techniques that can be used for deepfake detection and the advantages of each of them. First is the Analysis of biological singles, this technique focuses on the eye blinking of the person, it uses CNN and RNN combination to recognize physiological signals like blinking and eye movement. Next, it uses a binary classifier to see if the eyes of the person are open or closed. The next Method is the Analysis of Spatial and Temporal features, which basically uses a CNN to extract frame information which is then sent to an LSTM layer to analyze a time series to detect facial expression changes between consecutive images, then the video is categorized as real or fake. He also suggested using YOLO-CNN-XGBoost. This includes using YOLO for extracting the features from the video and the CNN and XGBoost to make the prediction, this model may achieve a 90% AUROC in receiver operating characteristic plots. Datasets used include CASIA-WebFace, DFFD, VGGFace2, 100K-Faces, FFHQ, The eye-blinking dataset, and DeepfakeTIMIT.

Rahman et.al [8] Have trained a convolutional neural network that has achieved good accuracy in low-resolution and short-time video data. The datasets used are the DFDC dataset and FaceForensics++.

In terms of pre-processing the video, the whole dataset video cropped frame by frame-by-frame facial image. DLIB was used to recognize the facial features in the video. Then the video is trained among 3 models to compare the models including InceptionResNetV2, MobileNet, and DenseNet121. In all the models pre-trained ImageNet weights were added to the first layer of every model. For compiling the models Adam optimizer with a learning rate of 1e-5, decay of 0.0, loss-function as Binary-cross-entropy, and matrix as accuracy. The split used for training the model was

70/20/10 for training, validation, and testing respectively. InceptionResNetV2 achieved an accuracy of 93.7%, MobileNet achieved an accuracy of 94.93% and DenseNet121 achieved an accuracy of 93.86%. Overall the DFDC achieved an accuracy of 94.93% and FaceForensics++ achieved an accuracy of 93.2%.

3. Dataset Description

The Celeb-DF dataset consists of 590 original videos collected from YouTube where the subjects of the video were of different ages, ethnic groups, and genders. The videos are mostly celebrity figures during interviews. In correspondence to these 590 videos, 5639 Deepfake videos were made. The deepfake videos in these datasets have various visual augmentations that can help distinguish them from the real videos. Some of the augmentation techniques that have been used are traditional augmentations, GridMask, face morphing, and style transfer.

4. Methodology

The main goal of the work is to identify if the video is real or a deep fake which has been generated through some technology. Since the input is a video and the Deep learning neural networks accept only images as input the videos need to be converted to images as shown in Fig. 1. Since in the Celeb-Df dataset most of the changes in the videos are in the facial features the full the body of the subject is not required. So the pre-processing process consists of 3 steps: detecting the face, checking if the video is corrupt, and extracting the frames from the video. Each of the steps is discussed in detail below.

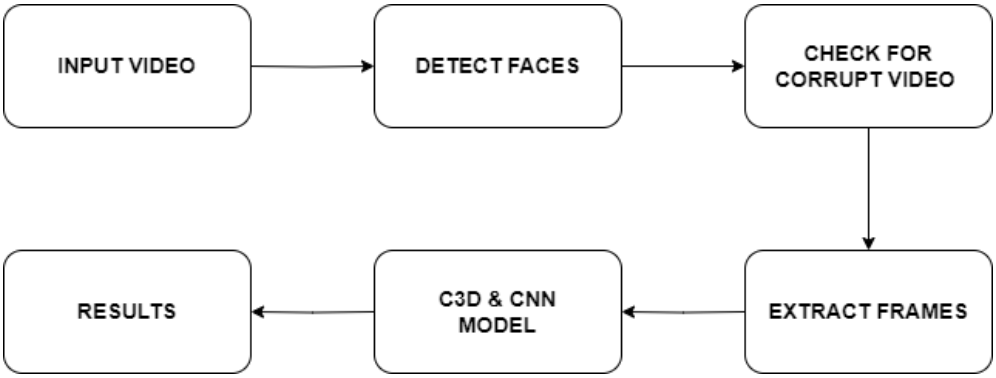

Figure 1 Methodology

First, from the input, the face from the video was detected and saved as a separate video. This was done using cv2 and the face_recognition module as shown in Fig 2. The final videos have a size of 112x112 pixels and the processing involved skipping every 4th frame and only the first 150 frames were considered.

```
def frame_extract(path):
  vidObj = cv2.VideoCapture(path)
  success = 1
  while success:
      success, image = vidObj.read()
      if success:
          yield image
import torch
import torchvision
from torchvision import transforms
from torch.utils.data import DataLoader
from torch.utils.data.dataset import Dataset
import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
import face_recognition
from tqdm import tqdm
#process the frames
def create_face_videos(path_list,out_dir):
  already_present_count =  glob.glob(out_dir+'*.mp4')
  print("No of videos already present " , len(already_present_count))
  for path in tqdm(path_list):
    out_path = os.path.join(out_dir,path.split('/')[-1])
    file_exists = glob.glob(out_path)
    if(len(file_exists) != 0):
      print("File Already exists: " , out_path)
      continue
    frames = []
    flag = 0
    face_all = []
    frames1 = []
    out = cv2.VideoWriter(out_path,cv2.VideoWriter_fourcc('M','J','P','G'), 30, (112,112))
    for idx,frame in enumerate(frame_extract(path)):
      #if(idx % 3 == 0):
      if(idx <= 150):
        frames.append(frame)
        if(len(frames) == 4):
          faces = face_recognition.batch_face_locations(frames)
          for i,face in enumerate(faces):
            if(len(face) != 0):
              top,right,bottom,left = face[0]
            try:
              out.write(cv2.resize(frames[i][top:bottom,left:right,:],(112,112)))
            except:
              pass
          frames = []
    try:
      del top,right,bottom,left
    except:
      pass
```

Figure 2 Code for extracting face from video

Second, the face videos were validated to make sure that the videos didn't corrupt when converting them from the input videos, so there were no issues when extracting the frames from the video as shown in Fig 3. A neural network based on a face recognition model was used to transform and validate the frames. If the video was corrupted it was removed from the dataset. During the validation process, a little transformation was done to prepare it for frame extraction. The transformation done was normalization, the pixel values were normalized.

```
[ ]  #THis code is to check if the video is corrupted or not..
     #If the video is corrupted delete the video.
     import glob
     import torch
     import torchvision
     from torchvision import transforms
     from torch.utils.data import DataLoader
     from torch.utils.data.dataset import Dataset
     import os
     import numpy as np
     import cv2
     import matplotlib.pyplot as plt
     import face_recognition
     #Check if the file is corrupted or not
     def validate_video(vid_path,train_transforms):
           transform = train_transforms
           count = 20
           video_path = vid_path
           frames = []
           a = int(100/count)
           first_frame = np.random.randint(0,a)
           temp_video = video_path.split('/')[-1]
           for i,frame in enumerate(frame_extract(video_path)):
             frames.append(transform(frame))
             if(len(frames) == count):
               break
           frames = torch.stack(frames)
           frames = frames[:count]
           return frames
     #extract a from from video
     def frame_extract(path):
       vidObj = cv2.VideoCapture(path)
       success = 1
       while success:
           success, image = vidObj.read()
           if success:
               yield image

     im_size = 112
     mean = [0.485, 0.456, 0.406]
     std = [0.229, 0.224, 0.225]

     train_transforms = transforms.Compose([
                                            transforms.ToPILImage(),
                                            transforms.Resize((im_size,im_size)),
                                            transforms.ToTensor(),
                                            transforms.Normalize(mean,std)])
     video_fil =  glob.glob('/content/drive/My Drive/DesignProject-Keane_Dr.Tamizharasan/Dataset/FF_REAL_Face_only_data/*.mp4')
     video_fil +=  glob.glob('/content/drive/My Drive/DesignProject-Keane_Dr.Tamizharasan/Dataset/FF_FAKE_Face_only_data/*.mp4')

     print("Total no of videos :" , len(video_fil))
     print(video_fil)
     count = 0;
     for i in tqdm(video_fil):
       try:
         count+=1
         validate_video(i,train_transforms)
       except:
         print("Number of video processed: " , count ," Remaining : " , (len(video_fil) - count))
         print("Corrupted video is : " , i)
         continue
     print((len(video_fil) - count))
```

Figure 3 Code to check if the video is corrupt

Third, the frames of all the videos were extracted and put into separate folders (individual folders for individual videos). Using Open CV to read the videos, extract individual frames, and save the JPEG images as shown in Fig 4 and 5. The videos were then put into a frames folder and then separated as 1 and 0, 1 containing the deep fake videos and 0 containing the real videos.

```
[ ]  from os import makedirs,path
     from tqdm import tqdm

     real_path='/content/drive/My Drive/DesignProject-Keane_Dr.Tamizharasan/Dataset/FF_REAL_Face_only_data/'
     deepfake_path='/content/drive/My Drive/DesignProject-Keane_Dr.Tamizharasan/Dataset/FF_FAKE_Face_only_data/'


     train_path = [real_path]
     for folder in train_path:
         videos_path = glob.glob(path.join(folder, "*.mp4"))
         folder = folder.split("/")[0:5]
         folder="/".join(folder)
         for video_path in tqdm(videos_path):
             cap = cv2.VideoCapture(video_path)
             vid = video_path.split("/")[-1]
             vid = vid.split(".")[0]
             frameRate = cap.get(5)  # frame rate

             if not path.exists('/content/frames/0'+"/video_" +vid):
                 makedirs('/content/frames/0'+"/video_" +vid)
             while cap.isOpened():
               frameId = cap.get(1)  # current frame number
               ret, frame = cap.read()
               if not ret:
                 break
               filename = (
                   '/content/frames/0'
                  +"/video_"
                  +vid
                  + "/image_"
                  + str(int(frameId) + 1)
                  +".jpg"
               )
               cv2.imwrite(filename, frame)
             cap.release()

     100%|██████████| 588/588 [00:50<00:00, 11.61it/s]
```

Figure 4 Code to convert Real videos to images

```
[ ]  from os import makedirs,path
     from tqdm import tqdm

     real_path='/content/drive/My Drive/DesignProject-Keane_Dr.Tamizharasan/Dataset/FF_REAL_Face_only_data/'
     deepfake_path='/content/drive/My Drive/DesignProject-Keane_Dr.Tamizharasan/Dataset/FF_FAKE_Face_only_data/'


     train_path = [deepfake_path]
     for folder in train_path:
         videos_path = glob.glob(path.join(folder, "*.mp4"))
         folder = folder.split("/")[0:5]
         folder="/".join(folder)
         for video_path in tqdm(videos_path):
             cap = cv2.VideoCapture(video_path)
             vid = video_path.split("/")[-1]
             vid = vid.split(".")[0]
             frameRate = cap.get(5)  # frame rate

             if not path.exists('/content/frames/1'+"/video_" +vid):
                 makedirs('/content/frames/1'+"/video_" +vid)
             while cap.isOpened():
               frameId = cap.get(1)  # current frame number
               ret, frame = cap.read()
               if not ret:
                 break
               filename = (
                   '/content/frames/1'
                   +"/video_"
                   +vid
                   + "/image_"
                   + str(int(frameId) + 1)
                   +".jpg"
               )
               cv2.imwrite(filename, frame)
             cap.release()

100%|██████████| 5709/5709 [08:35<00:00, 11.08it/s]
```

Figure 5 Code to convert Fake videos to images

4.1 C3D Neural Network

The proposed architecture is a C3D neural network as shown in Fig 6. The model begins with a layer that represents a 5-dimensional tensor with the shape (batch_size, 112, 112,3) where 112 represents the width and height of the video and 3 is the RGB color channels. The network consists of five layers, each consisting of a 3D convolutional layer followed by a max pooling layer. The Convolution layer has a kernel size of 3x3x3 and a Rectified Linear Unit (ReLU) activation, this layer is used to extract spatial features from the frames of the video. Max pooling layers are used to downsample the spatial dimensions of the feature maps, enhancing the computational efficiency. After

the convolutional layer, the architecture transitions to fully connected layers, which include 2 fully connected layers of 2048 units each and a ReLU activation. To avoid overfitting, dropout layers are added between the fully connected layers with a dropout rate of 0.5. The final layer is a dense layer with 2 units which is used for the binary classification of whether it is a deep fake or real, the dense layer uses softmax activation, giving probability scores for each class. The main reason for using C3D is so that it would be able to capture both spatial and temporal information.

4.2 CNN XceptionNet model

The model is initialized by loading the Xception base model with pre-trained weights from ImageNet. Then additional layers are added to the base model for fine-tuning. A global average pooling layer is added which is used to reduce the spatial dimensions of the feature maps to a single value per channel by taking the average. This operation aids in the spatially invariant capture of the most important features, resulting in a sort of spatial compression. A dense layer with 512 units and ReLU activation is added to the architecture, to extract the high-level features. A dropout layer with a dropout rate of 0.4 is applied to prevent overfitting. After that, a dropout layer with a rate of 0.5 is added to further increase the model's generalization capability. The final layer consists of a dense layer with two units and softmax activation. The entire model is then compiled using the Nadam optimizer with a learning rate of 0.002 using categorical cross-entropy loss.

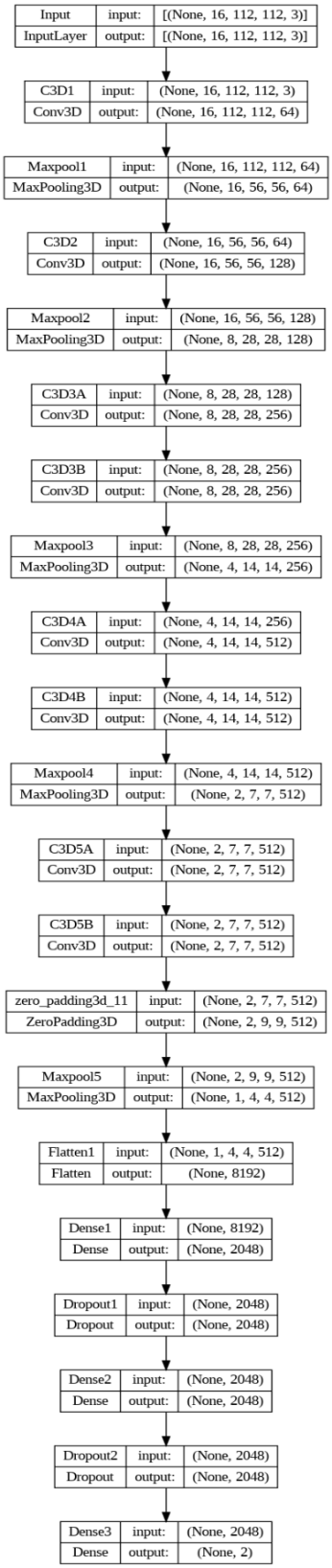

Figure 6 C3D Model

5. Implementation

5.1 Training Configuration

The training configuration for the neural network starts with dividing the dataset into training and validation sets. 80% of the videos are used for training and about 20% of the videos are designated for validation. For C3D neural network training process is conducted with a batch size of 16, meaning 16 videos are used for each iteration. The training is conducted over 15 epochs. Two main optimizers used were Adam optimizer and SGD, these optimizers were used with different learning rates of 0.001, 0.005, and 0.01. Different optimizers were used to see which one helped minimize the categorical cross-entropy loss.

The CNN training process was also conducted with a batch size of 16, The training was conducted over 10 epochs. The optimizer used was Nadam and the learning rate was 0.002.

5.2 Experimental Setup

In our experiment, the minimum requirements are Windows 10 or more as an operating system and Google Colab as an integrated development environment (IDE). The modules used were keras 2.15.0, numpy 1.23.5, and cv2 4.8.0. GPU is essential for this machine-learning project appropriate versions of Tensorflow, CUDA, and cuDNN need to be installed.

5.3 Evaluation Metrics

Even though neural networks have been used it is still binary classification hence the model has been tested on accuracy and loss. The loss function used is categorical cross-entropy.

***Accuracy score*:** Accuracy is the proportion of correct predictions made by a machine learning model over total predictions.

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

***Categorical cross-entropy*:** Also known as Softmax Loss. It is used to quantify the dissimilarity between the predicted probabilities and the true categorical labels.

$$L = -\frac{1}{N} \sum_{i=1}^{N} = \sum_{j=1}^{C} = y_{ij}\, log(p_{ij})$$

6. Results

Fig 7 and 8 show the C3D model results, providing us with a thorough look at how the model performed under its best optimization options. It was discovered that the Adam Optimizer with a learning rate of 0.01 worked best, demonstrating the model's ability to train efficiently. Now, looking at Fig 7, we can see how the model's accuracy changed during training and validation. The highest validation accuracy was 50%, providing a good indication of how effectively the model generalizes. On the training side, the average accuracy settled at a dependable 52%, indicating that the model learned from the training data consistently throughout multiple epochs. This in-depth examination of accuracy trends not only highlights the model's prediction abilities but also illustrates the significance of evaluating both training and validation metrics for a thorough evaluation.

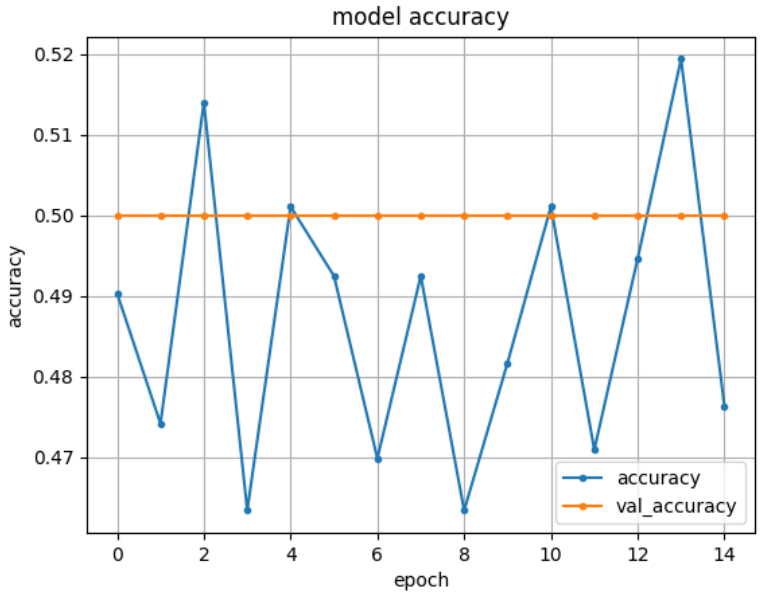

Figure 7 C3D Model Accuracy

Fig 8 illustrates a comprehensive picture of the model's loss landscape, covering the complexity of training as well as the variations of validation. This visual tour of the model's performance metrics provides a full understanding of how it dealt with the dataset's complexities. When we look at Figure 8, we notice that there were losses during both the training and validation periods. There are two sides to this story: The average validation loss stayed constant at 0.6933, revealing the model's ability to estimate and make reasonable predictions on new, previously unknown data. Meanwhile, the mean loss during training was 0.8131, indicating that the model can learn from the training set, albeit with a little greater loss than during validation.

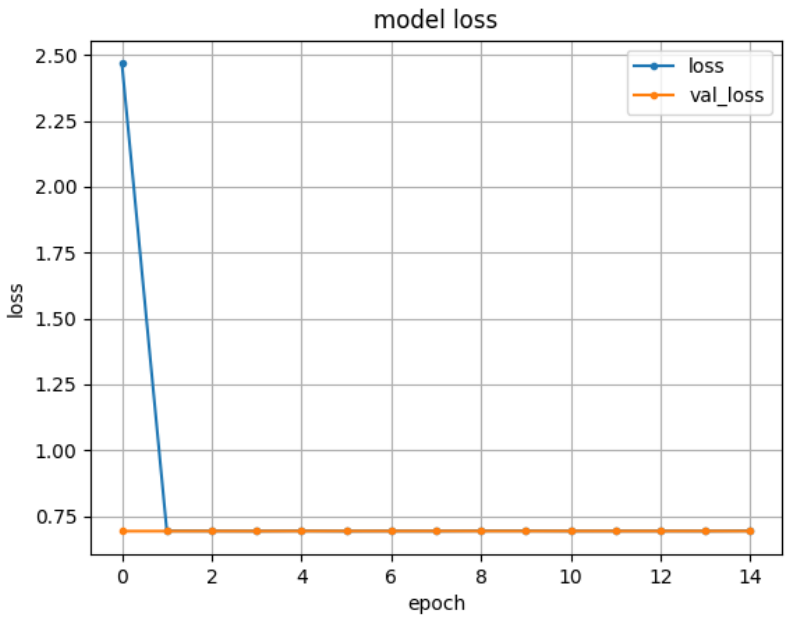

Figure 8 C3D Model Loss

Examining the results produced by our CNN model, the graphic depiction provided in Fig 9 and 10 provides an in-depth account of the model's operation in various optimization settings. After a thorough investigation of different optimization setups, the Nadam Optimizer, along with a carefully chosen learning rate of 0.002, proved to be the most effective combo for obtaining the most impressive outcomes. As a visual tour guide across the accuracy landscape of the model, Fig 9 offers a nuanced perspective by displaying both the important validation accuracy and the dynamic growth of training accuracy. Interestingly, the validation accuracy peaked at a remarkable 78%, demonstrating the model's ability to generalize well to previously encountered data. In terms of training, the model demonstrated its aptitude for learning, as seen by the highest training accuracy reaching a strong 94%.

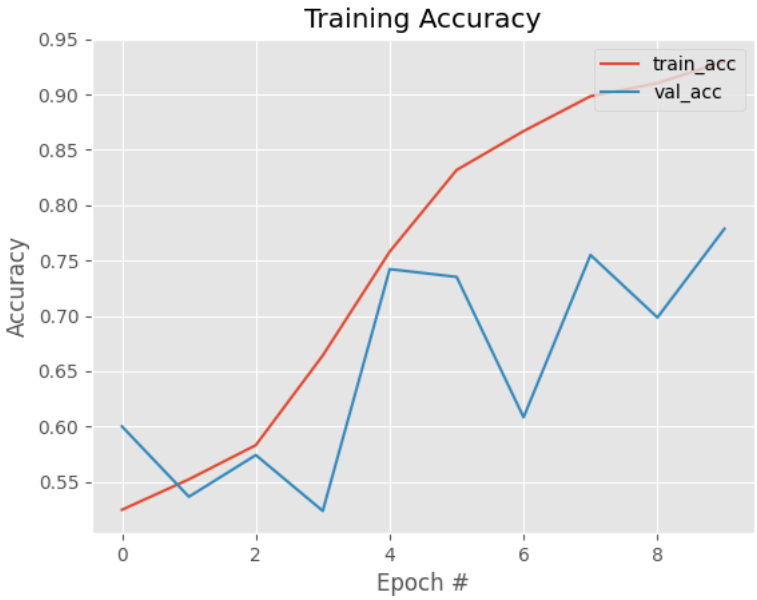

Figure 9 CNN Model Accuracy

Fig 10 focuses on the loss trends during the training and validation process in particular. Zooming into the training phase, we were able to get a very low loss value of 0.2382. It appears as though the model breezed through its training process, demonstrating an amazing capacity to reduce mistakes and identify trends from the training set. Our model continued to demonstrate its capabilities during the validation phase, with the lowest loss value reaching 0.6814. This demonstrates the model's ability to generalize well beyond the observed data, upholding an impressive performance level even in the face of novel, unforeseen cases.

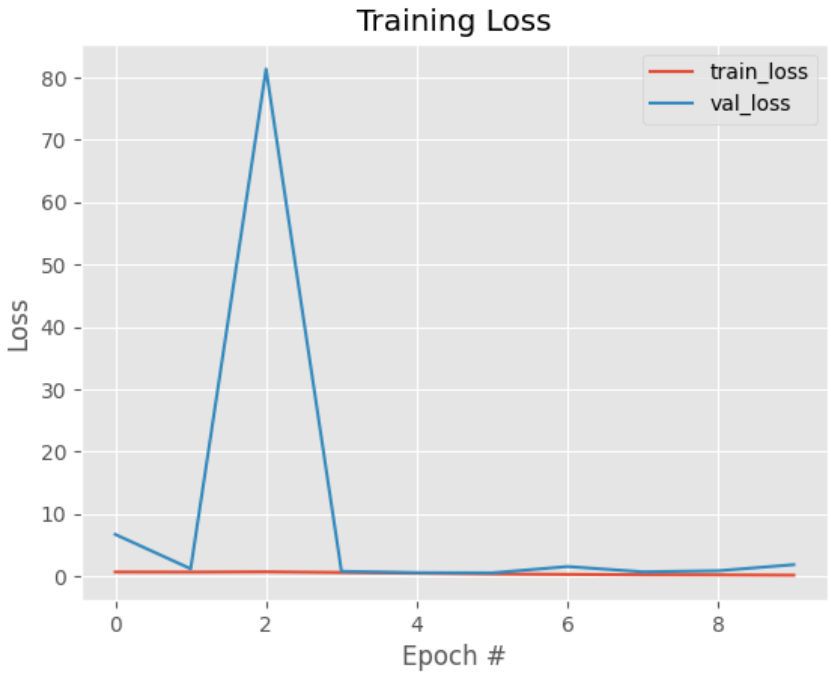

Figure 10 CNN Model Loss

7. Conclusion & Future Scope

In this paper we used neural networks for the classification of deepfakes one was using C3D and was a CNN using Xception Net. The C3D was used since we assumed that it would be able to capture the facial features much better than other models. The validation accuracy was 50% which might not be the best but the model's training accuracy went up to 52 % which shows that the model was learning. The CNN model performed much better than the C3D model with a validation accuracy of 78% and a training accuracy of 94 %. The accuracy of the CNN could have been increased but the training data size had to be reduced to GPU and RAM constraints. As mentioned previously the C3D model was tested with different optimizers and learning rates and the difference in results was key. From this, we can conclude that architecture and optimization play a key role in neural networks.

There are many more options to consider as we proceed in order to improve the performance of both models. A potential direction is to refine the C3D model by carefully examining its architecture. Improve the model's ability to identify complex spatiotemporal features, this entails adjusting the sizes of the layers, possibly improving their arrangements, and thinking about adding dropout layers. It's similar to giving the model a minor makeover to improve its overall performance. Increasing the size of our training dataset can also be expected to yield a significant improvement in performance since it will enable the models to extract more subtle and insightful information from a wider range of samples.Ultimately, more varied data frequently leads to more reliable and broadly applicable models. Including attention mechanisms in both models is an interesting way to step up the feature extraction game. By doing this, we enable the models to concentrate on the most important features in the video data, which may improve their capacity to recognize and understand complex dynamics and patterns. To put it simply, the key to maximizing the models' potential in the field of video analysis is to strategically combine architectural modifications, data augmentation, and the addition of attention mechanisms.

REFERENCES

[1] Rana, M.S. *et al.* (2022) 'Deepfake detection: A systematic literature review', *IEEE Access*, 10, pp. 25494–25513. doi:10.1109/access.2022.3154404.

[2] Shahroz Tariq, Sangyup Lee, Simon S. Woo, 'A convolutional LSTM based Residual Network for deep fake video detection', doi: 10.48550/arXiv.2009.07480

[3] Zhao, H. *et al.* (2021) 'Multi-attentional deepfake detection', *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* [Preprint]. doi:10.1109/cvpr46437.2021.00222.

[4] Roy, R. *et al.* (2022) '3D CNN architectures and attention mechanisms for deepfake detection', *Handbook of Digital Face Manipulation and Detection*, pp. 213–234. doi:10.1007/978-3-030-87664-7_10.

[5] Guera, D. and Delp, E.J. (2018) 'Deepfake video detection using recurrent neural networks', *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)* [Preprint]. doi:10.1109/avss.2018.8639163.

[6] Chang, X. *et al.* (2020) 'Deepfake face image detection based on improved VGG convolutional neural network', *2020 39th Chinese Control Conference (CCC)* [Preprint]. doi:10.23919/ccc50068.2020.9189596.

[7] Chintha, A. *et al.* (2020) 'Recurrent convolutional structures for audio spoof and Video deepfake detection', *IEEE Journal of Selected Topics in Signal Processing*, 14(5), pp. 1024–1037. doi:10.1109/jstsp.2020.2999185.

[8] Mira, F. (2023) 'Deep Learning Technique for recognition of Deep fake videos', *2023 IEEE IAS Global Conference on Emerging Technologies (GlobConET)* [Preprint]. doi:10.1109/globconet56651.2023.10150143.

[9] Rahman, A. *et al.* (2022) 'Short and low-resolution Deepfake video detection using CNN', *2022 IEEE 10th Region 10 Humanitarian Technology Conference (R10-HTC)* [Preprint]. doi:10.1109/r10-htc54060.2022.9929719.

Deepfake Detection using deep learning

ORIGINALITY REPORT

6%	1%	6%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS