RAJALAKSHMI ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

DEEPFAKE DETECTION (FAKE VISION)

Submitted by

LOKESH G 221501066
MANIKANDAN K 221501074

AI19541 FUNDAMENTALS OF DEEP LEARNING

Department of Artificial Intelligence and Machine Learning

Rajalakshmi Engineering College, Thandalam

RAJALAKSHMI ENGINEERING COLLEGE

BONAFIDE CERTIFICATE

NAME …………………………………………………………………………………..……..…

ACADEMIC YEAR………………..………SEMESTER…………..BRANCH………………

UNIVERSITY REGISTER No.

Certified that this is the bonafide record of work done by the above students in the Mini Project titled **"DEEPFAKE DETECTION (FAKE VISION)"** in the subject **AI19541 – FUNDAMENTALS OF DEEP LEARNING** during the year **2024 - 2025.**

Signature of Faculty – in – Charge

Submitted for the Practical Examination held on ______________________

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT

The growing computation power has made the deep learning algorithms so powerful that creating a indistinguishable human synthesized video popularly called as deep fakes have became very simple. Scenarios where these realistic face swapped deep fakes are used to create political distress, fake terrorism events, revenge porn, blackmail peoples are easily envisioned. In this work, we describe a new deep learning-based method that can effectively distinguish AI-generated fake videos from real videos.Our method is capable of automatically detecting the replacement and reenactment deep fakes. We are trying to use Artificial Intelligence(AI) to fight Artificial Intelligence(AI). Our system uses a Res-Next Convolution neural network to extract the frame-level features and these features and further used to train the Long Short Term Memory(LSTM) based Recurrent Neural Network(RNN) to classify whether the video is subject to any kind of manipulation or not, i.e whether the video is deep fake or real video. To emulate the real time scenarios and make the model perform better on real time data, we evaluate our method on large amount of balanced and mixed data-set prepared by mixing the various available data-set like Face-Forensic++[1], Deepfake detection challenge[2], and Celeb-DF[3]. We also show how our system can achieve competitive result using very simple and robust approach.

Keywords:

Res-Next Convolution neural network.

Recurrent Neural Network (RNN).

Long Short Term Memory(LSTM).

Computer vision

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

PROJECT IDEA

In the world of ever growing Social media platforms, Deepfakes are considered as the major threat of the AI. There are many Scenarios where these realistic face swapped deepfakes are used to create political distress, fake terrorism events, revenge porn, blackmail peoples are easily envisioned. Some of the examples are Brad Pitt, Angelina Jolie nude videos.

It becomes very important to spot the difference between the deepfake and pristine video. We are using AI to fight AI. Deepfakes are created using tools like FaceApp[11] and Face Swap [12], which using pre-trained neural networks like GAN or Auto encoders for these deepfakes creation. Our method uses a LSTM based artificial neural network to process the sequential temporal analysis of the video frames and pre-trained Res-Next CNN to extract the frame level features. ResNext Convolution neural network extracts the frame-level features and these features are further used to train the Long Short Term Memory based artificial Recurrent Neural Network to classify the video as Deepfake or real. To emulate the real time scenarios and make the model perform better on real time data, we trained our method with large amount of balanced and combination of various available dataset like FaceForensic++[1], Deepfake detection challenge[2], and Celeb-DF[3].

Further to make the ready to use for the customers, we have developed a front end application where the user the user will upload the video. The video will be processed by the model and the output will be rendered back to the user with the classification of the video as deepfake or real and confidence of the model.

CHAPTER 2
LITERATURE REVIEW

1. Face Warping Artifacts

Face Warping Artifacts [15] used the approach to detect artifacts by comparing the generated face areas and their surrounding regions with a dedicated Convolutional Neural Networkmodel. In this work there were two-fold of Face Artifacts.Their method is based on the observations that current deepfake algorithm can only generate images of limited resolutions, which are then needed to be further transformed to match the faces to be replaced in thesource video. Their method has not considered the temporal analysis of the frames.

2.Detection using Eye Blinking

Detection by Eye Blinking [16] describes a new method for detecting the deepfakes by the eye blinking as a crucial parameter leading to classification of the videos as deepfake or pristine. The Long-term Recurrent Convolution Network (LRCN) was used for temporal analysis of the cropped frames of eye blinking. As today the deepfake generation algorithms have become so powerful that lack of eye blinking can not be the only clue for detection of the deepfakes. There must be certain other parameters must be considered for the detection of deepfakes like teeth enchantment, wrinkles on faces, wrong placement of eyebrows etc.

3.Forges image detection using capsule networks

Capsule networks to detect forged images and videos [17] uses a method that uses a capsule network to detect forged, manipulated images and videos in different scenarios, like replay attack detection and computer-generated video detection. In their method, they have used random noise in the training phase which is not a good option. Still the model performed beneficial in their dataset but may fail on real time data due to noise in training. Our method is proposed to be trained on noiseless and real time datasets. Recurrent Neural Network [18] (RNN) for deepfake detection used the approach of using RNN for sequential processing of the frames along with ImageNet pre-trained model. Their process used the HOHO [19] dataset consisting of just 600 videos.Their dataset consists small number of videos and same type of videos, which may not perform very well on the real time data. We will be training out model on large number of Realtime data.

4.Systhentic portrait videos using biological signals

Synthetic Portrait Videos using Biological Signals [20] approach extract biological signals from facial regions on pristine and deepfake portrait video pairs. Applied transformations to compute the spatial coherence and temporal consistency, capture the signal characteristics in feature vector and photoplethysmography (PPG) maps, and further train a probabilistic Support Vector Machine (SVM) and a Convolutional Neural Network (CNN). Then, the average of authenticity probabilities is used to classify whether the video is a deepfake or a pristine.

Fake Catcher detects fake content with high accuracy, independent of the generator, content, resolution, and quality of the video. Due to lack of discriminator leading to the loss in their findings to preserve biological signals, formulating a differentiable loss function that follows the proposed signal processing steps is not straight forward process.

CHAPTER 3

SYSTEM REQUIREMENTS

3.1 HARDWARE REQUIREMENTS:

- Processor: Intel Core i5/Ryzen 5 minimum
- RAM: 8 GB minimum (16 GB recommended)
- GPU: NVIDIA GPU (e.g., GTX 1050 or better)
- Storage: 10 GB free space

3.2 SOFTWARE REQUIRED:

- Operating System: Windows 10/11, macOS, or Linux
- Python: Version 3.8 or higher
- GIT: For version control.
- Jupyter Notebook/ VSCode: For Python code testing and debugging.
- Database: SQLite or Firebase
- Flutter SDK: Latest stable version
- Android Studio or VS Code: For Flutter development and testing.
- Libraries: OpenCV, Numpy, Pandas, TensorFlow or PyTorch, Face recognition Library

CHAPTER 4

SYSTEM OVERVIEW

4.1 EXISTING SYSTEM

Existing deepfake detection systems face several limitations, including the need for extensive data and high computational power, which can make them resource-intensive and inaccessible for broader use. They often produce inconsistent results, lack advanced features, and struggle with variations in lighting and motion consistency. Additionally, these systems are computationally expensive, limiting their effectiveness in real-time applications. Our proposed system addresses these challenges by using a ResNeXt50 model for efficient feature extraction and an LSTM network for analyzing temporal inconsistencies in videos, optimizing performance even with limited computational resources. Furthermore, it incorporates real-time processing capabilities, robust classification, and intuitive result visualization, ensuring reliable and user-friendly deepfake detection.

4.2 PROPOSED SYSTEM

The proposed system enables users to upload videos or images via the app for deepfake detection. It employs a pre-trained Haar Cascade Classifier for face detection and cropping, followed by feature extraction using a ResNeXt50 model to capture relevant facial details. For videos, an LSTM network analyzes frame sequences to identify inconsistencies characteristic of deepfakes. A fully connected layer then classifies the input as REAL or FAKE, with results visualized alongside confidence scores and heatmaps highlighting key areas. The system is deployed with a Flask backend that processes inputs and returns results, seamlessly integrated with a Flutter app for user interaction.

4.2.1 SYSTEM ARCHITECTURE

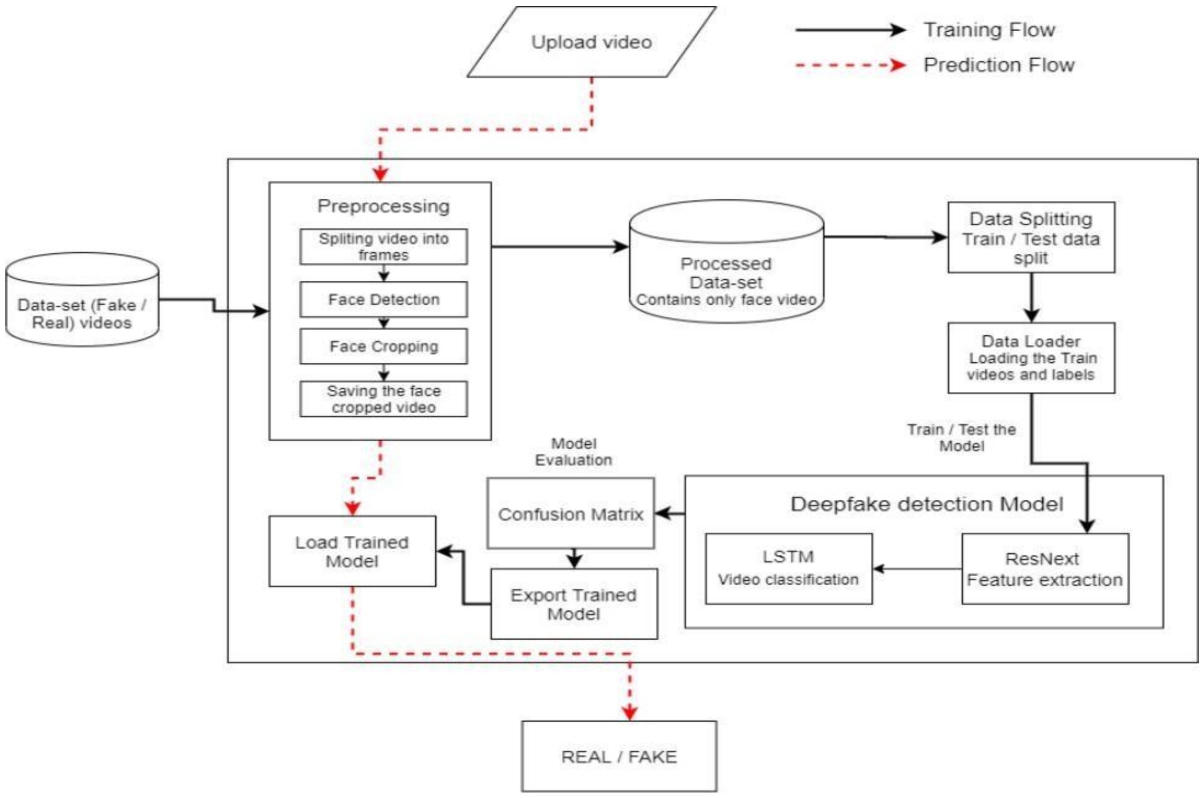

Upload video
Training Flow
Prediction Flow
Preprocessing
Spliting video into frames
Face Detection
Face Cropping
Saving the face cropped video
Data-set (Fake / Real) videos
Processed Data-set Contains only face video
Data Splitting Train / Test data split
Data Loader Loading the Train videos and labels
Train / Test the Model
Model Evaluation
Confusion Matrix
Deepfake detection Model
LSTM Video classification
ResNext Feature extraction
Load Trained Model
Export Trained Model
REAL / FAKE

Fig 1.1 Overall diagram

4.2.2 DESCRIPTION

In this system, we have trained our PyTorch deepfake detection model on equal number of real and fake videos in order to avoid the bias in the model. The system architecture of the model is showed in the figure. In the development phase, we have taken a dataset, preprocessed the dataset and created a new processed dataset which only includes the face cropped videos. In essence, the diagram outlines a streamlined process for epilepsy detection. It begins with acquiring and refining EEG data, then proceeds to train and evaluate SVM and Random Forest models. Finally, integration into a real-time monitoring system ensures prompt identification of seizures, facilitating timely medical response.

4.2.1.1 SYSTEM FLOW

The system flow begins with the user uploading a video, which is then passed through a pre-processing stage involving face detection and cropping using a pre-trained Haar Cascade Classifier. This step ensures that only the relevant facial regions are considered for analysis. The processed video frames are then fed into a trained deep learning model.

The model utilizes a ResNeXt50-based CNN for feature extraction and an LSTM-based RNN to analyze temporal inconsistencies in videos. Based on the extracted features, the system classifies the input video as either "REAL" or "FAKE" and displays the result to the user. This streamlined approach ensures efficient and accurate detection of deepfake videos while maintaining the ease of use.

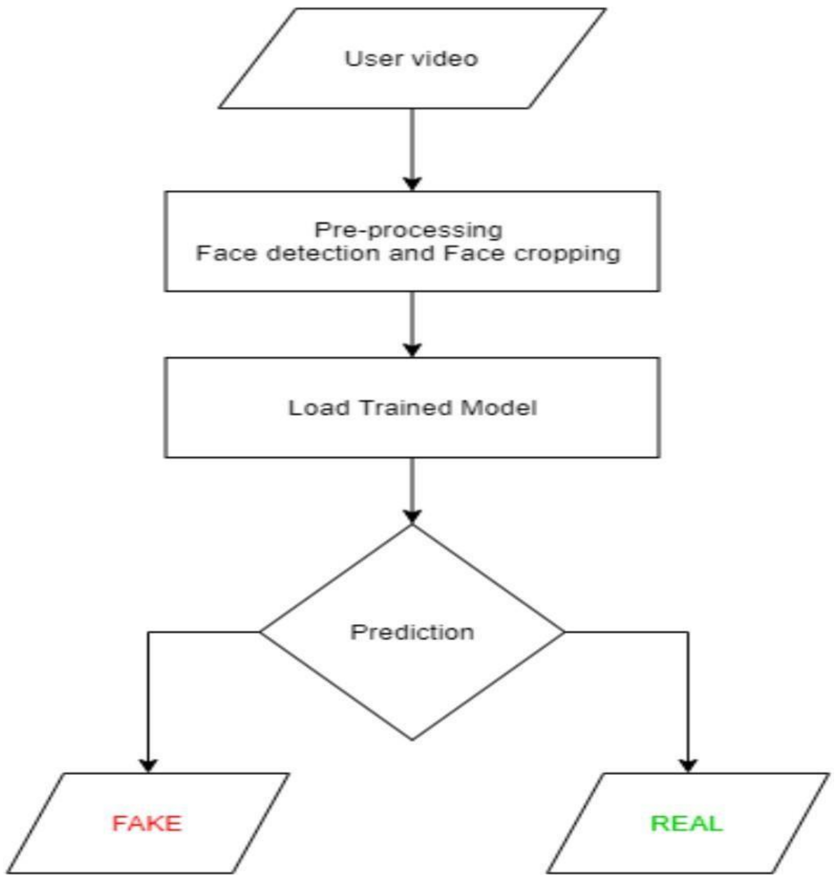


Fig 1.2 System Flow Diagram

CHAPTER-5

IMPLEMENTATION

5.1 LIST OF MODULES

- Data-set Gathering
- Data Pre processing
- Data-set split
- Model Architecture
- Hyper-Parameter tuning

5.2 MODULE DESCRIPTION

5.2.1 Module 1 : Data-set Gathering

For making the model efficient for real time prediction. We have gathered the data from different available data-sets like FaceForensic++(FF)[1], Deepfake detection challenge(DFDC)[2], and Celeb-DF[3]. Futher we have mixed the dataset the collected datasets and created our own new dataset, to accurate and real time detection on different kind of videos. To avoid the training bias of the model we have considered 50% Real and 50% fake videos.

Deep fake detection challenge (DFDC) dataset [3] consist of certain audio alerted video, asaudio deepfake are out of scope for this paper. We preprocessed the DFDC dataset and removed the audio altered videos from the dataset by running a python script.

After preprocessing of the DFDC dataset, we have taken 1500 Real and 1500 Fake videos from the DFDC dataset. 1000 Real and 1000 Fake videos from the FaceForensic++(FF)[1] dataset and 500 Real and 500 Fake videos from the CelebDF[3] dataset. Which makes our total dataset consisting 3000 Real, 3000 fake videos and 6000 videos in total. Figure 2 depicts the distribution of the data-sets.

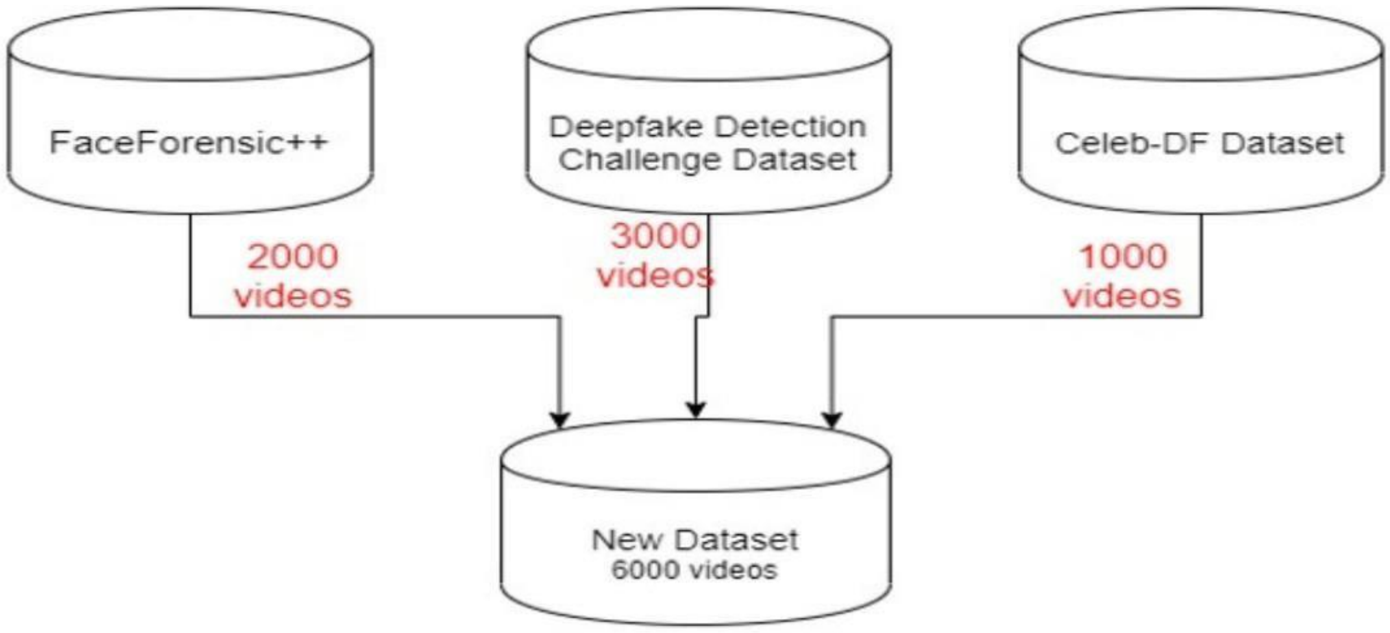
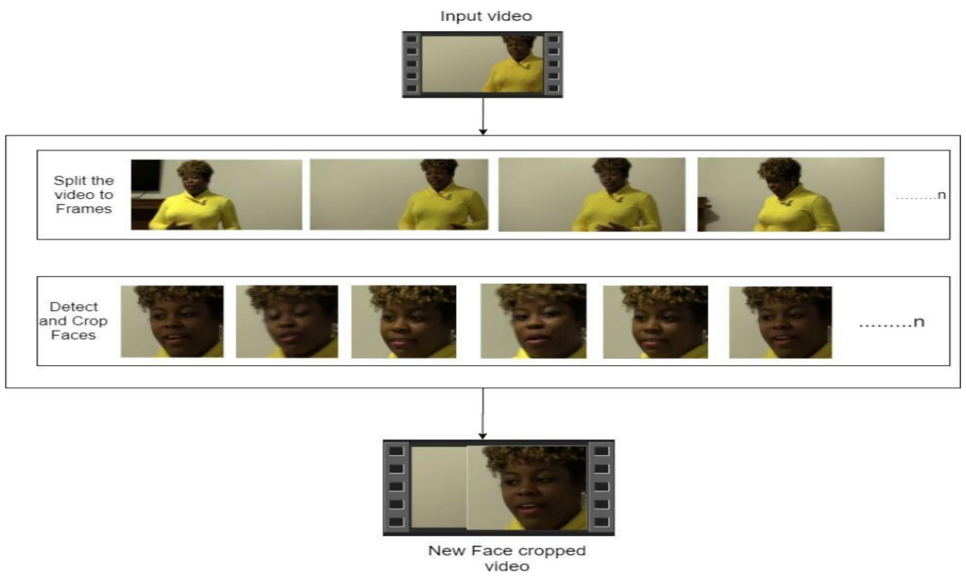


Fig 5.2.1 Dataset Gathering

5.2.2 Module 2 : Pre-processing

In this step, the videos are preprocessed and all the unrequired and noise is removed from videos. Only the required portion of the video i.e face is detected and cropped. The first steps in the preprocessing of the video is to split the video into frames. After splitting the video into frames the face is detected in each of the frame and the frame is cropped along the face. Later the cropped frame is again converted to a new video by combining each frame of the video. The process is followed for each video which leads to creation of processed dataset containing face only videos. The frame that does not contain the face is ignored while preprocessing. To maintain the uniformity of number of frames, we have selected a threshold value based on the mean of total frames count of each video. Another reason for selecting a threshold value is limited computation power. As a video of 10 second at 30 frames per second(fps) will have total 300 frames and it is computationally very difficult to process the 300 frames at a single time in the experimental environment. So, based on our Graphic Processing Unit (GPU) computational power in experimental environment we have selected 150 frames as the threshold value. While saving the frames to the new dataset we have only saved the first 150 frames of the video to the new video. To demonstrate the proper use of Long Short-Term Memory (LSTM) we have considered the frames in the sequential manner i.e. first 150 frames and not randomly. The newly created video is saved at frame rate of 30 fps and resolution of 112 x 112.



5.2.3 Module 3: Data-set split

The dataset is split into train and test dataset with a ratio of 70% train videos (4,200) and 30% (1,800) test videos. The train and test split is a balanced split i.e 50% of the real and 50% of fake videos in each split.

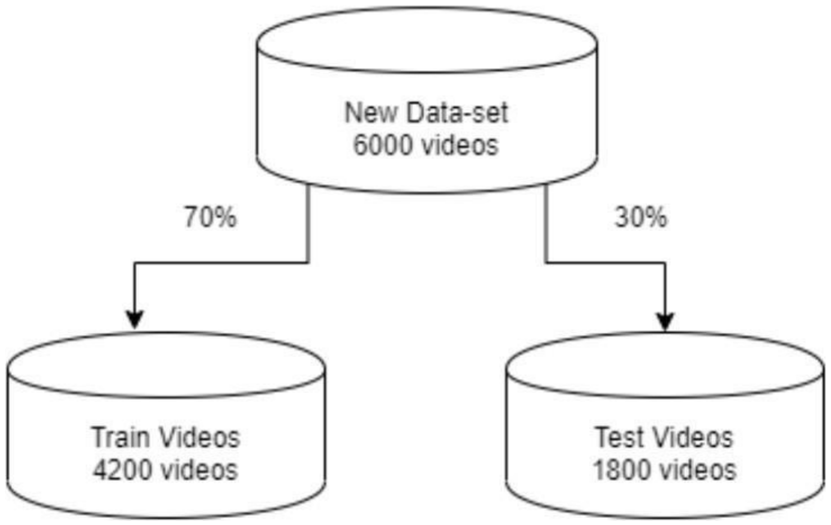


Fig 5.2.3 Data-Split Figure

5.2.4 Module 4: Model Architecture

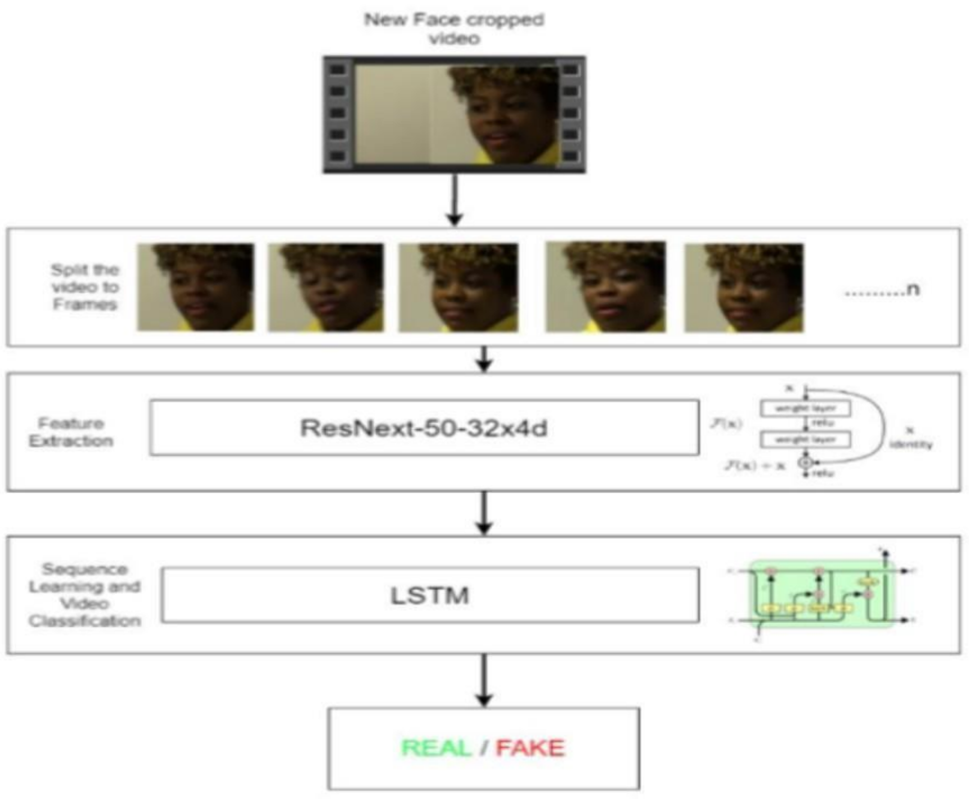
Our model is a combination of CNN and RNN. We have used the Pre- trained ResNext CNN model to extract the features at frame level and based on the extracted features a LSTM network is trained to classify the video as deepfake or pristine. Using the Data Loader on training split of videos the labels of the videos are loaded and fitted into the model for training.

ResNext :

Instead of writing the code from scratch, we used the pre-trained model of ResNext for feature extraction. ResNext is Residual CNN network optimized for high performance on deeper neural networks. For the experimental purpose we have used resnext50_32x4d model. We have used a ResNext of 50 layers and 32 x 4 dimensions. Following, we will be fine-tuning the network by adding extra required layers and selecting a proper learning rate to properly converge the gradient descent of the model. The 2048-dimensional feature vectors after the last pooling layers of ResNext is used as the sequential LSTM input.

LSTM for Sequence Processing:

2048-dimensional feature vectors is fitted as the input to the LSTM. We are using 1 LSTM layer with 2048 latent dimensions and 2048 hidden layers along with 0.4 chance of dropout, which is capable to do achieve our objective. LSTM is used to process the frames in a sequential manner so that the temporal analysis of the video can be made, by comparing the frame at ‘t’ second with the frame of ‘t-n’ seconds. Where n can be any number of frames before t. The model also consists of Leaky Relu activation function. A linear layer of 2048 input features and 2 output features are used to make the model capable of learning the average rate of correlation between eh input and output. An adaptive average polling layer with the output parameter 1 is used in the model. Which gives the the target output size of the image of the form H x W. For sequential processing of the frames a Sequential Layer is used. The batch size of 4 is used to perform the batch training. A SoftMax layer is used to get the confidence of the model during predication.



5.2.5 Module 5: Hyper-parameter tuning

It is the process of choosing the perfect hyper-parameters for achieving the maximum accuracy. After reiterating many times on the model. The best hyper-parameters for our dataset are chosen. To enable the adaptive learning rate Adam[21] optimizer with the model parameters is used. The learning rate is tuned to 1e-5 (0.00001) to achieve a better global minimum of gradient descent. The weight decay used is 1e-3.

As this is a classification problem so to calculate the loss cross entropy approach is used.To use the available computation power properly the batch training is used. The batch size is taken of 4. Batch size of 4 is tested to be ideal size for training in our development environment.

The User Interface for the application is developed using Flutter Frontend framework. Firebase is used to enable the scalability of the application in the future.

CHAPTER-6

RESULT AND DISCUSSION

Our system effectively detects and classifies deepfake videos, achieving high accuracy in distinguishing manipulated content from authentic videos. By leveraging a robust deep learning framework, including ResNeXt CNN and LSTM-based RNN, the model demonstrates competitive performance on diverse datasets such as FaceForensics++, Deepfake Detection Challenge, and Celeb-DF. The user-friendly interface ensures seamless video analysis, making it accessible for real-world applications to combat deepfake misuse.

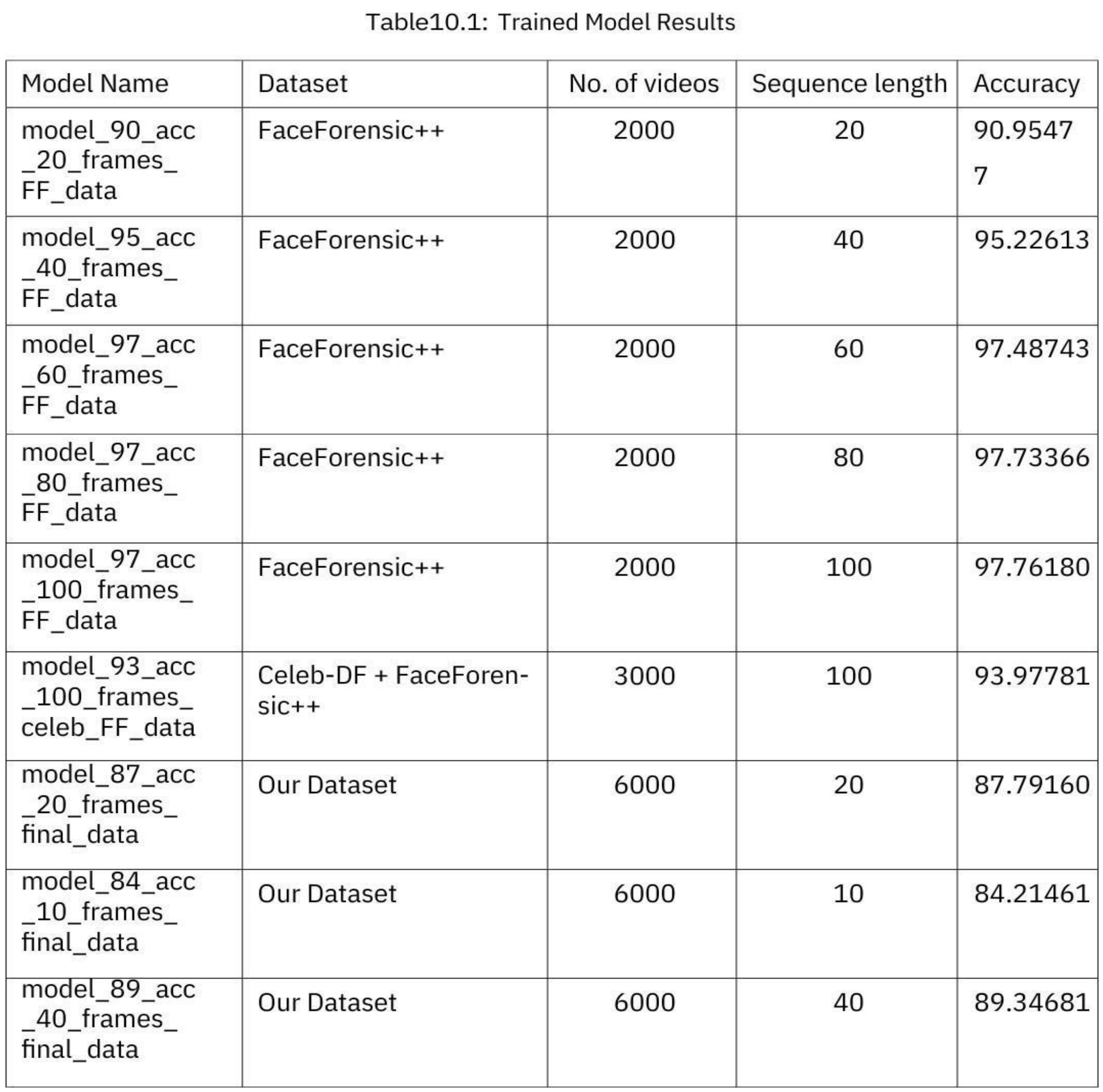


Table10.1: Trained Model Results

Model Name	Dataset	No. of videos	Sequence length	Accuracy
model_90_acc_20_frames_FF_data	FaceForensic++	2000	20	90.95477
model_95_acc_40_frames_FF_data	FaceForensic++	2000	40	95.22613
model_97_acc_60_frames_FF_data	FaceForensic++	2000	60	97.48743
model_97_acc_80_frames_FF_data	FaceForensic++	2000	80	97.73366
model_97_acc_100_frames_FF_data	FaceForensic++	2000	100	97.76180
model_93_acc_100_frames_celeb_FF_data	Celeb-DF + FaceForensic++	3000	100	93.97781
model_87_acc_20_frames_final_data	Our Dataset	6000	20	87.79160
model_84_acc_10_frames_final_data	Our Dataset	6000	10	84.21461
model_89_acc_40_frames_final_data	Our Dataset	6000	40	89.34681

Fig 6.1 Output Screenshot of the Deepfake Detection Model

REFERENCE

[1] Andreas Rossler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies,Matthias Nießner, “FaceForensics++: Learning to Detect Manipulated Facial Images” in arXiv:1901.08971.

[2] Yuezun Li , Xin Yang , Pu Sun , Honggang Qi and Siwei Lyu “Celeb-DF: A Large-scale Challenging Dataset for DeepFake Forensics” in arXiv:1909.12962

[3] Deepfake Video of Mark Zuckerberg Goes Viral on Eve of House A.I. Hearing : https://fortune.com/2019/06/12/deepfake-mark-zuckerberg/ Accessed on 26 March, 2020

[4] G. Antipov, M. Baccouche, and J.-L. Dugelay. Face aging with conditional generative adversarial networks. arXiv:1702.01983, Feb. 2017

[5] J. Thies et al. Face2Face: Real-time face capture and reenactment of rgb videos. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2387–2395, June 2016. Las Vegas, NV.

[6] Face app: https://www.faceapp.com/ (Accessed on 26 March, 2020)

[7] Face Swap : https://faceswaponline.com/ (Accessed on 26 March, 2020)

APPENDIX

SAMPLE CODE

```
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


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

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/Celeb_fake_face_only/*.mp4')
video_fil += glob.glob('/content/drive/My Drive/Celeb_real_face_only/*.mp4')
video_fil += glob.glob('/content/drive/My Drive/DFDC_FAKE_Face_only_data/*.mp4')
video_fil += glob.glob('/content/drive/My Drive/DFDC_REAL_Face_only_data/*.mp4')
video_fil += glob.glob('/content/drive/My Drive/FF_Face_only_data/*.mp4')
print("Total no of videos :" , len(video_fil))
print(video_fil)
count = 0;
for i in 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))

import json
import glob
import numpy as np
import cv2
import copy
import random
video_files =  glob.glob('/content/drive/My Drive/Celeb_fake_face_only/*.mp4')
video_files += glob.glob('/content/drive/My Drive/Celeb_real_face_only/*.mp4')
video_files += glob.glob('/content/drive/My Drive/DFDC_FAKE_Face_only_data/*.mp4')
video_files += glob.glob('/content/drive/My Drive/DFDC_REAL_Face_only_data/*.mp4')
video_files += glob.glob('/content/drive/My Drive/FF_Face_only_data/*.mp4')
random.shuffle(video_files)
random.shuffle(video_files)
```

```
frame_count = []
for video_file in video_files:
  cap = cv2.VideoCapture(video_file)
  if(int(cap.get(cv2.CAP_PROP_FRAME_COUNT))<100):
  video_files.remove(video_file)
    continue
frame_count.append(int(cap.get(cv2.CAP_PROP_FRAME_COUNT)))
print("frames are " , frame_count)
print("Total no of video: " , len(frame_count))
print('Average frame per video:',np.mean(frame_count))

# load the video name and labels from csv
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
class video_dataset(Dataset):
    def _init_(self,video_names,labels,sequence_length = 60,transform = None):
        self.video_names = video_names
        self.labels = labels
        self.transform = transform
        self.count = sequence_length
    def _len_(self):
        return len(self.video_names)
    def _getitem_(self,idx):
        video_path = self.video_names[idx]
        frames = []
        a = int(100/self.count)
        first_frame  =  np.random.randint(0,a)
        temp_video  =  video_path.split('/')[-1]
        #print(temp_video)
        label = self.labels.iloc[(labels.loc[labels["file"] == temp_video].index.values[0]),1]
        if(label == 'FAKE'):
          label = 0
        if(label == 'REAL'):
          label = 1
        for i,frame in enumerate(self.frame_extract(video_path)):
          frames.append(self.transform(frame))
          if(len(frames) == self.count):
            break
        frames = torch.stack(frames)
        frames = frames[:self.count]
        #print("length:" , len(frames), "label",label)
        return frames,label
    def frame_extract(self,path):
      vidObj = cv2.VideoCapture(path)
      success = 1
      while success:
          success, image = vidObj.read()
          if success:
              yield image
#plot the image
def im_plot(tensor):
    image = tensor.cpu().numpy().transpose(1,2,0)
    b,g,r = cv2.split(image)
    image = cv2.merge((r,g,b))
    image = image*[0.22803, 0.22145, 0.216989] +  [0.43216, 0.394666, 0.37645]
    image = image*255.0
    plt.imshow(image.astype(int))
    plt.show()

#count the number of fake and real videos
def number_of_real_and_fake_videos(data_list):
  header_list = ["file","label"]
  lab = pd.read_csv('/content/drive/My Drive/Gobal_metadata.csv',names=header_list)
  fake = 0
  real = 0
  for i in data_list:
    temp_video = i.split('/')[-1]
    label = lab.iloc[(labels.loc[labels["file"] == temp_video].index.values[0]),1]
    if(label == 'FAKE'):
      fake+=1
    if(label == 'REAL'):
      real+=1
  return real,fake
```

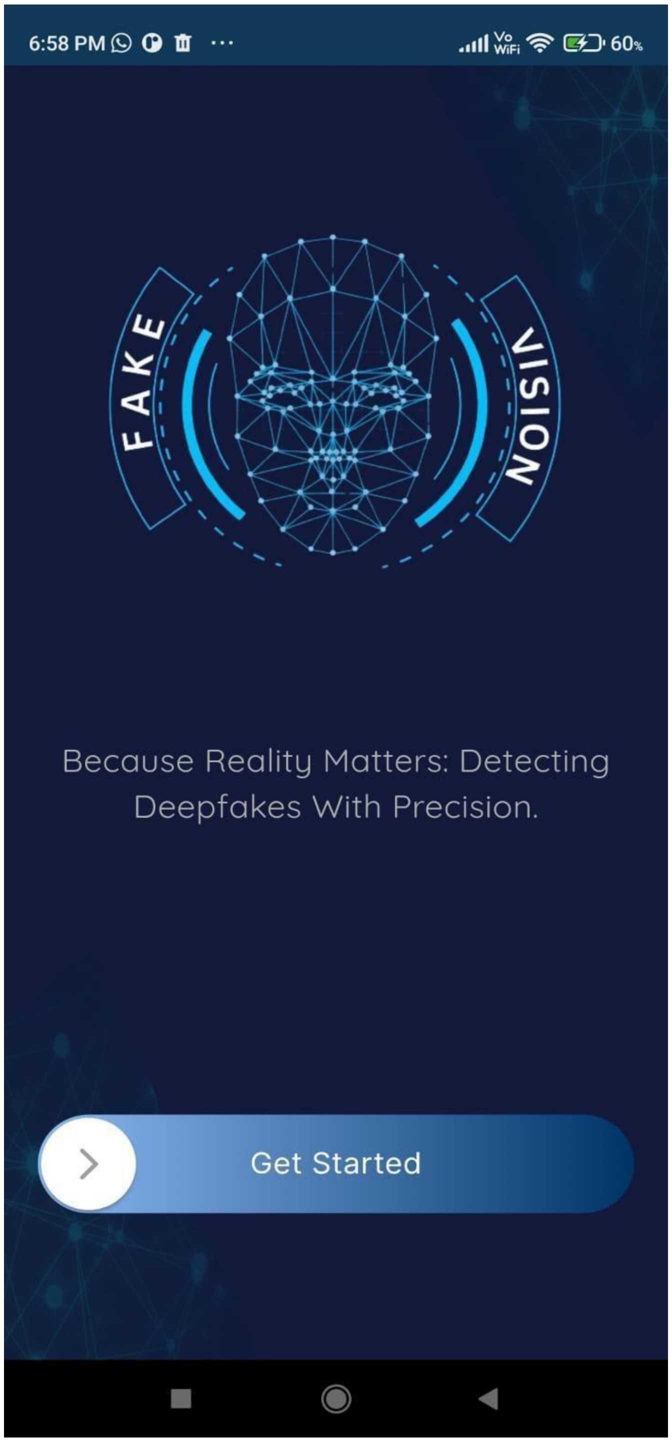
```
# load the labels and video in data loader
import random
import pandas as pd
from sklearn.model_selection import train_test_split

header_list = ["file","label"]
labels = pd.read_csv('/content/drive/My Drive/Gobal_metadata.csv',names=header_list)
#print(labels)
train_videos = video_files[:int(0.8*len(video_files))]
valid_videos = video_files[int(0.8*len(video_files)):]
print("train : " , len(train_videos))
print("test : " , len(valid_videos))
# train_videos,valid_videos = train_test_split(data,test_size = 0.2)
# print(train_videos)

print("TRAIN: ", "Real:",number_of_real_and_fake_videos(train_videos)[0],"
Fake:",number_of_real_and_fake_videos(train_videos)[1])
print("TEST: ", "Real:",number_of_real_and_fake_videos(valid_videos)[0],"
Fake:",number_of_real_and_fake_videos(valid_videos)[1])


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)])

test_transforms = transforms.Compose([
                                        transforms.ToPILImage(),
                                        transforms.Resize((im_size,im_size)),
                                        transforms.ToTensor(),
                                        transforms.Normalize(mean,std)])
train_data = video_dataset(train_videos,labels,sequence_length = 10,transform =
train_transforms)
#print(train_data)
val_data = video_dataset(valid_videos,labels,sequence_length = 10,transform =
train_transforms)
```

```
train_loader = DataLoader(train_data,batch_size = 4,shuffle = True,num_workers = 4)
valid_loader = DataLoader(val_data,batch_size = 4,shuffle = True,num_workers = 4)
image,label = train_data[0]
im_plot(image[0,:,:,:])

#Model with feature visualization
from torch import nn
from torchvision import models
class Model(nn.Module):
   def _init_(self, num_classes,latent_dim= 2048, lstm_layers=1 , hidden_dim = 2048,
bidirectional = False):
      super(Model, self)._init_()
      model = models.resnext50_32x4d(pretrained = True) #Residual Network CNN
      self.model = nn.Sequential(*list(model.children())[:-2])
      self.lstm = nn.LSTM(latent_dim,hidden_dim, lstm_layers,  bidirectional)
      self.relu = nn.LeakyReLU()
      self.dp = nn.Dropout(0.4)
      self.linear1 = nn.Linear(2048,num_classes)
      self.avgpool = nn.AdaptiveAvgPool2d(1)
   def forward(self, x):
      batch_size,seq_length, c, h, w = x.shape
      x = x.view(batch_size * seq_length, c, h, w)
      fmap = self.model(x)
      x = self.avgpool(fmap)
      x = x.view(batch_size,seq_length,2048)
      x_lstm,_ = self.lstm(x,None)
      return fmap,self.dp(self.linear1(torch.mean(x_lstm,dim = 1)))

model = Model(2).cuda()
a,b = model(torch.from_numpy(np.empty((1,20,3,112,112))).type(torch.cuda.FloatTensor))

import torch
from torch.autograd import Variable
import time
import os
import sys
import os
def train_epoch(epoch, num_epochs, data_loader, model, criterion, optimizer):
   model.train()
   losses = AverageMeter()
   accuracies = AverageMeter()
```

```
    t = []
    for i, (inputs, targets) in enumerate(data_loader):
        if torch.cuda.is_available():
            targets = targets.type(torch.cuda.LongTensor)
            inputs = inputs.cuda()
        _,outputs = model(inputs)
        loss  = criterion(outputs,targets.type(torch.cuda.LongTensor))
        acc = calculate_accuracy(outputs, targets.type(torch.cuda.LongTensor))
        losses.update(loss.item(),  inputs.size(0))
        accuracies.update(acc,  inputs.size(0))
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        sys.stdout.write(
                "\r[Epoch %d/%d] [Batch %d / %d] [Loss: %f, Acc: %.2f%%]"
                % (
                    epoch,
                    num_epochs,
                    i,
                    len(data_loader),
                    losses.avg,
                    accuracies.avg))
    torch.save(model.state_dict(),'/content/checkpoint.pt')
    return losses.avg,accuracies.avg
def test(epoch,model, data_loader ,criterion):
    print('Testing')
    model.eval()
    losses = AverageMeter()
    accuracies = AverageMeter()
    pred = []
    true = []
    count = 0
    with torch.no_grad():
        for i, (inputs, targets) in enumerate(data_loader):
            if torch.cuda.is_available():
                targets = targets.cuda().type(torch.cuda.FloatTensor)
                inputs = inputs.cuda()
            _,outputs = model(inputs)
            loss  =  torch.mean(criterion(outputs, targets.type(torch.cuda.LongTensor)))
            acc = calculate_accuracy(outputs,targets.type(torch.cuda.LongTensor))
            _,p = torch.max(outputs,1)
            true +=
(targets.type(torch.cuda.LongTensor)).detach().cpu().numpy().reshape(len(targets)).tolist()
            pred += p.detach().cpu().numpy().reshape(len(p)).tolist()
            losses.update(loss.item(),  inputs.size(0))
            accuracies.update(acc,  inputs.size(0))
            sys.stdout.write(
                "\r[Batch %d / %d]  [Loss: %f, Acc: %.2f%%]"
                % (
                  i,
                  len(data_loader),
                  losses.avg,
                  accuracies.avg
                  )
                )
        print('\nAccuracy {}'.format(accuracies.avg))
    return true,pred,losses.avg,accuracies.avg
class AverageMeter(object):
    """Computes and stores the average and current value"""
    def _init_(self):
        self.reset()
    def reset(self):
        self.val = 0
        self.avg = 0
        self.sum = 0
        self.count = 0

    def update(self, val, n=1):
        self.val = val
        self.sum += val * n
        self.count += n
        self.avg = self.sum / self.count
def calculate_accuracy(outputs, targets):
    batch_size = targets.size(0)

    _, pred = outputs.topk(1, 1, True)
    pred = pred.t()
    correct = pred.eq(targets.view(1, -1))
    n_correct_elems = correct.float().sum().item()
    return 100* n_correct_elems / batch_size

def plot_loss(train_loss_avg,test_loss_avg,num_epochs):
  loss_train = train_loss_avg
  loss_val = test_loss_avg
  print(num_epochs)
  epochs = range(1,num_epochs+1)
  plt.plot(epochs, loss_train, 'g', label='Training loss')
  plt.plot(epochs, loss_val, 'b', label='validation loss')
  plt.title('Training and Validation loss')
  plt.xlabel('Epochs')
  plt.ylabel('Loss')
  plt.legend()
  plt.show()
def plot_accuracy(train_accuracy,test_accuracy,num_epochs):
  loss_train = train_accuracy
  loss_val = test_accuracy
  epochs = range(1,num_epochs+1)
  plt.plot(epochs, loss_train, 'g', label='Training accuracy')
  plt.plot(epochs, loss_val, 'b', label='validation accuracy')
  plt.title('Training and Validation accuracy')
  plt.xlabel('Epochs')
  plt.ylabel('Accuracy')
  plt.legend()
  plt.show()

from sklearn.metrics import confusion_matrix
#learning rate
lr = 1e-5#0.001
#number of epochs
num_epochs = 20

optimizer = torch.optim.Adam(model.parameters(), lr= lr,weight_decay = 1e-5)

#class_weights = torch.from_numpy(np.asarray([1,15])).type(torch.FloatTensor).cuda()
#criterion = nn.CrossEntropyLoss(weight = class_weights).cuda()
criterion = nn.CrossEntropyLoss().cuda()
train_loss_avg =[]
train_accuracy = []
test_loss_avg = []
test_accuracy = []
for epoch in range(1,num_epochs+1):
    l, acc = train_epoch(epoch,num_epochs,train_loader,model,criterion,optimizer)
    train_loss_avg.append(l)
    train_accuracy.append(acc)
    true,pred,tl,t_acc = test(epoch,model,valid_loader,criterion)
    test_loss_avg.append(tl)
    test_accuracy.append(t_acc)
plot_loss(train_loss_avg,test_loss_avg,len(train_loss_avg))
plot_accuracy(train_accuracy,test_accuracy,len(train_accuracy))
print(confusion_matrix(true,pred))
print_confusion_matrix(true,pred)
```

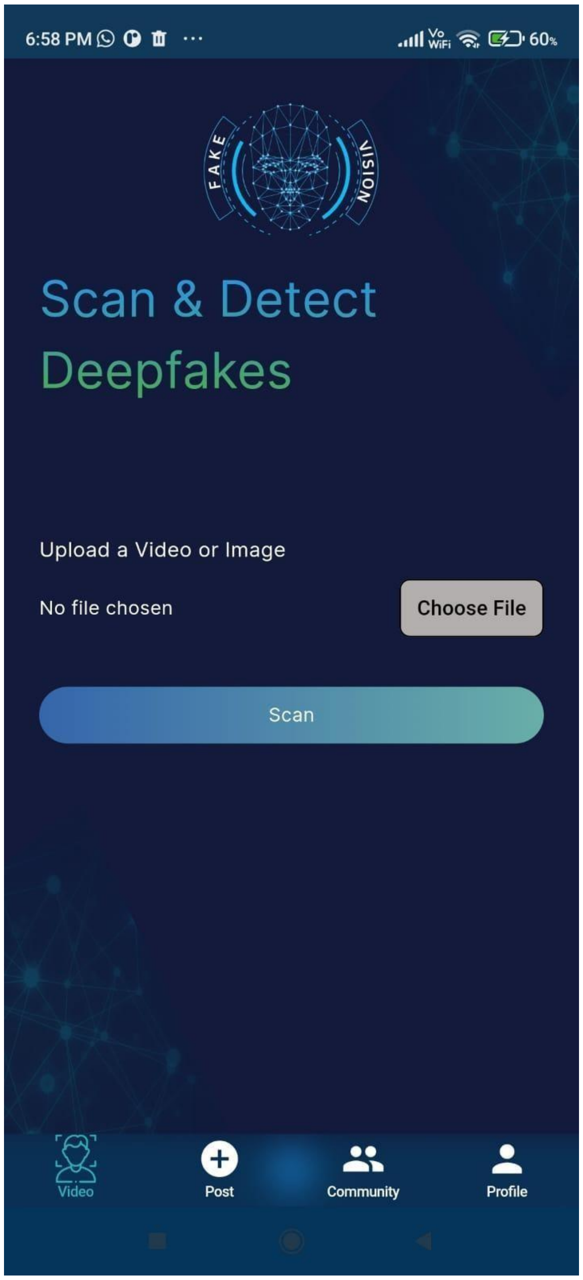
OUTPUT SCREENSHOTS

Login Page:

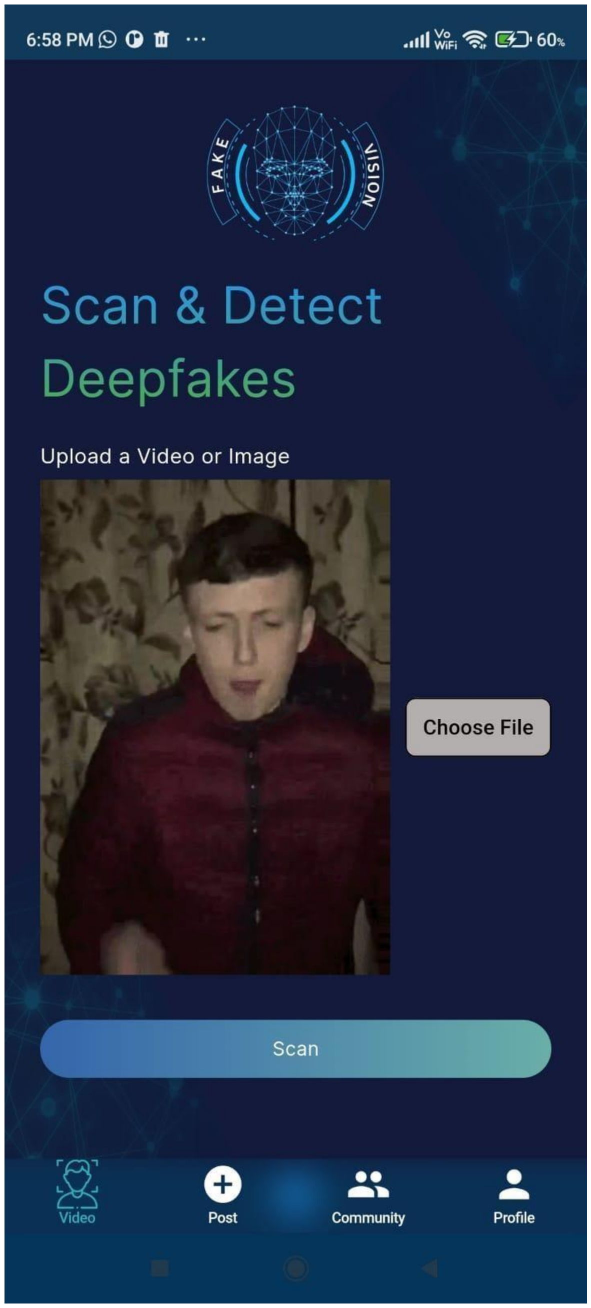


Because Reality Matters: Detecting Deepfakes With Precision.

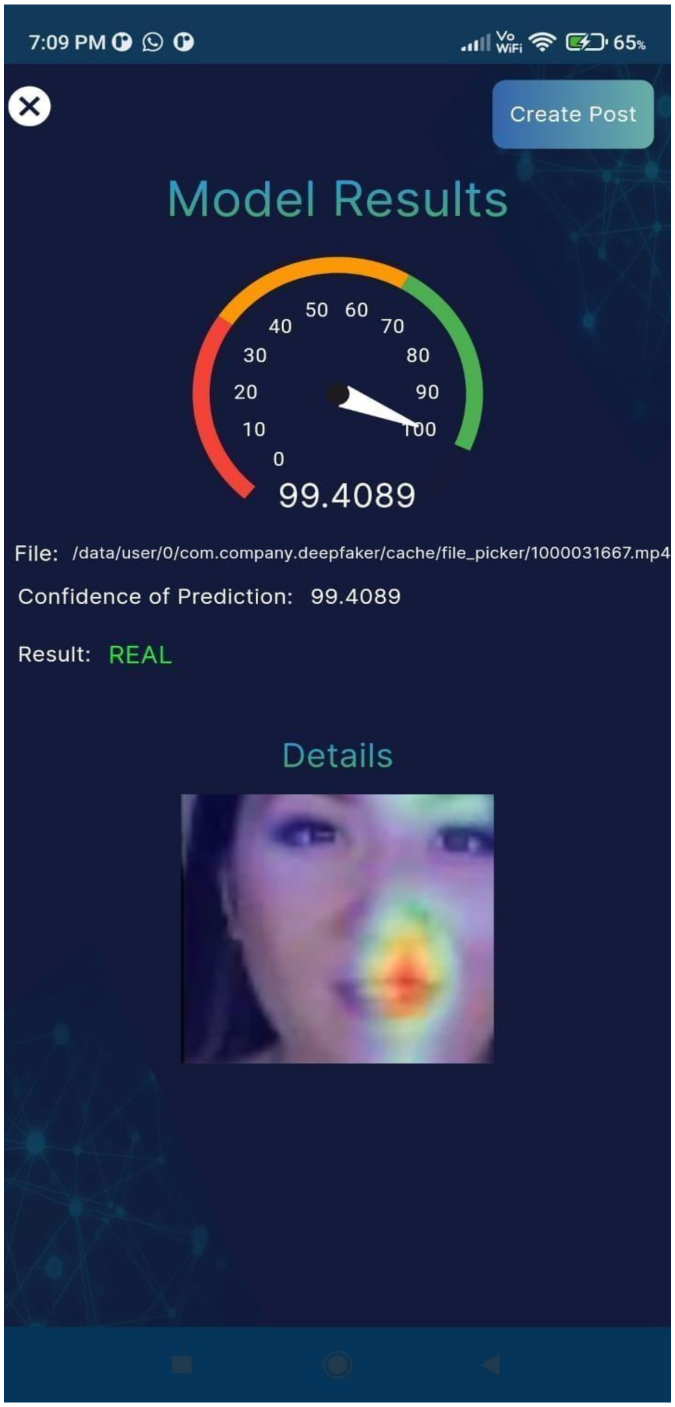
Home page:


6:58 PM
60%
FAKE VISION
Scan & Detect
Deepfakes
Upload a Video or Image
No file chosen
Choose File
Scan
Video
Post
Community
Profile

Deepfake detection page :



Result Page :



File: /data/user/0/com.company.deepfaker/cache/file_picker/1000031667.mp4

Confidence of Prediction: 99.4089

Result: REAL

FAKE VISION: DEEPFAKE DETECTION USING RESNET AND LSTM

Lokesh G
dept. Artificial Intelligence and Machine Learning
Rajalakshmi Engineering College
Chennai, India
221501065@rajalakshmi.edu.in

Sangeetha K
dept. Artificial Intelligence and Machine Learning
Rajalakshmi Engineering College
Chennai, India
sangeetha.k@rajalakshmi.edu.in

Manikandan K
dept. Artificial Intelligence and Machine Learning
Rajalakshmi Engineering College
Chennai, India
221501074@rajalakshmi.edu.in

Abstract—The rapid advancements in computational power and deep learning have simplified the creation of indistinguishable AI-synthesized videos, commonly known as deepfakes. These hyper-realistic videos have been misused to create political unrest, fake terrorism events, revenge porn, and blackmail scenarios, posing significant societal threats. This paper presents a novel deep learning-based approach to effectively distinguish deepfake videos from authentic ones. Our system leverages a ResNeXt-based Convolutional Neural Network (CNN) for extracting frame-level features, which are further processed using a Long Short-Term Memory (LSTM) Recurrent Neural Network (RNN) for classification. To address real-time challenges, the model is evaluated on a diverse dataset comprising FaceForensics++, Deepfake Detection Challenge, and Celeb-DF. The proposed method demonstrates robust and competitive performance with a simple yet effective approach, aiming to mitigate the misuse of deepfake technology and enhance detection reliability in real-world scenarios.

Keywords—Res-Next Convolution neural network. Recurrent Neural Network (RNN). Long Short Term Memory(LSTM). Computer vision, Deep learning.

I.INTRODUCTION

The rapid growth of social media platforms has brought numerous advancements, but it has also highlighted deepfakes as one of the major threats posed by Artificial Intelligence (AI). Realistic face-swapped deepfakes have been misused in various scenarios, such as creating political unrest, fake terrorism events, revenge porn, and blackmailing. Prominent examples include manipulated videos featuring celebrities like Brad Pitt and Angelina Jolie. Identifying the difference between deepfake and authentic videos has become crucial. To combat this threat, we propose leveraging AI to counteract AI. Deepfakes are commonly created using tools like FaceApp and FaceSwap, which utilize pre-trained neural networks such as Generative Adversarial Networks (GANs) or Autoencoders. Our method incorporates a Long Short-Term Memory (LSTM) artificial neural network for sequential temporal analysis of video frames and a pre-trained ResNeXt Convolutional Neural Network (CNN) for frame-level feature extraction.

II.RELATED WORK

Several methods have been proposed for deepfake detection, each with its own strengths and limitations. The Face Warping Artifacts method [15] focuses on detecting artifacts by comparing generated face areas with their surrounding regions using a dedicated Convolutional Neural Network. However, it does not consider temporal analysis, as it only analyzes static frames. The Detection by Eye Blinking method [16] utilizes eye blinking patterns as a crucial parameter for classification, employing Long-term Recurrent Convolution Networks (LRCN) for temporal analysis of cropped eye frames. While effective initially, this method is limited by advancements in deepfake algorithms that now accurately replicate natural blinking, necessitating additional parameters like facial wrinkles or eyebrow placement for better detection. Capsule Networks [17] have also been explored for detecting forged videos by leveraging random noise during training, but this approach may fail in real-world scenarios due to inconsistencies caused by noise.

III.PROBLEM STATEMENT

Convincing manipulations of digital images and videos have been demonstrated for several decades through the use of visual effects, recent advances in deep learning have led to a dramatic increase in the realism of fake content and the accessibility in which it can be created. These so-called AI-synthesized media (popularly referred to as deep fakes).Creating the Deep Fakes using the Artificially intelligent tools are simple task. But, when it comes to detection of these Deep Fakes, it is major challenge. Already in the history there are many examples where the deepfakes are used as powerful way to create political tension[14], fake terrorism events, revenge porn, blackmail peoples etc. So it becomes very important to detect these deepfake and avoid the percolation of deepfake through social media platforms. We have taken a step forward in detecting the deep fakes using LSTM based artificial Neural network.

IV.SYSTEM ARCHITECTURE AND DESIGN

The proposed deepfake detection system is built on a robust architecture designed to effectively distinguish between real and fake videos. The model, developed in PyTorch, is trained on an equal number of real and fake videos to eliminate any bias, ensuring balanced and accurate detection. During the development phase, the dataset undergoes preprocessing, where videos are split into frames, and only face-cropped frames are retained to create a processed dataset focused solely on relevant facial regions. This preprocessing step enhances the model's ability to identify artifacts unique to deepfakes.
Understanding the deepfake creation process is crucial for its detection. Tools like GANs and autoencoders typically use a source image and a target video, splitting the video into frames, replacing the source face with the target face, and combining the altered frames using pre-trained models. These tools also refine the video quality by removing residual traces, making deepfakes appear highly realistic.

V.PROPOSED METHODOLOGY

The proposed system enables users to upload videos or images via the app for deepfake detection. It employs a pre-trained Haar Cascade Classifier for face detection and cropping, followed by feature extraction using a ResNeXt50 model to capture relevant facial details. For videos, an LSTM network analyzes frame sequences to identify inconsistencies characteristic of deepfakes. A fully connected layer then classifies the input as REAL or FAKE, with results visualized alongside confidence scores and heatmaps highlighting key areas. The system is deployed with a Flask backend that processes inputs and returns results, seamlessly integrated with a Flutter app for user interaction.

To classify the input as REAL or FAKE, the extracted features are passed through a fully connected layer, which generates predictions along with confidence scores. The results are presented to the user through visualizations, including heatmaps that highlight manipulated areas, providing clarity and insight into the detection process. The Flask backend ensures efficient processing and seamless communication with the app, enabling real-time detection and feedback.

VI.IMPLEMENTATION AND RESULTS

The proliferation of deepfake technology has led to the widespread creation of misleading and manipulative content on social media platforms, often targeting famous personalities. Examples include deepfake videos of Mark Zuckerberg during the House A.I. Hearing, Donald Trump as James McGill in a "Breaking Bad" parody, and Barack Obama’s public service announcement, among others [5]. These deepfakes create panic and confusion, highlighting the urgent need for accurate detection systems to distinguish them from authentic videos. Advances in modern deep learning frameworks such as TensorFlow, Keras, and PyTorch, coupled with accessible high computational power, have revolutionized video manipulation. Conventional autoencoders and Generative Adversarial Networks (GANs) have enabled the creation of realistic and tampered videos with ease. Applications like FaceApp and Face Swap further simplify deepfake generation, offering features such as facial transformations, hairstyle changes, and age or gender modifications. While these tools have legitimate uses, they have also been extensively exploited for malicious purposes, such as fake celebrity pornographic videos, revenge porn [13], fake surveillance videos, and political misinformation. High-profile targets, including celebrities and politicians, are frequently victimized by such realistic fake content. Consequently, detecting deepfake videos and preventing their spread on social media platforms has become imperative to mitigate their harmful impacts.

VII.CONCLUSION AND FUTURE WORK

We presented a neural network-based approach to classify the video as deep fake or real, along with the confidence of proposed model. Our method is capable of predicting the output by processing 1 second of video (10 frames per second) with a good accuracy. We implemented the model by using pre-trained ResNext CNN model to extract the frame level features and LSTM for temporal sequence processing to spot the changes between the t and t-1 frame. Our model can process the video in the frame sequence of 10,20,40,60,80,100.

Any system, especially one built using the latest technological advancements, always offers scope for improvement and scalability. For this deepfake detection system, future enhancements could include developing a browser plugin to provide users with seamless and easy access to detection tools directly within their web browsers. Additionally, while the current algorithm is focused on detecting face-based deepfakes, there is significant potential to expand its capabilities to identify full-body deepfakes. Such enhancements would broaden the system’s applicability and further strengthen its utility in combating the misuse of deepfake technology.