FAKE VIDEO CLASSIFICATION USING LSTM

MAJOR PROJECT STAGE-II REPORT

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

BY

AMENA TOUSIATH UNNISA --------(20JJ1A1203)

Under the Guidance Of

Dr. S. SURESH KUMAR

Assistant Professor & Head of **IT**

DEPARTMENT OF INFORMATION TECHNOLOGY

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY

HYDERABAD UNIVERSITY COLLEGE OF ENGINEERING JAGTIAL

Nachupally (Kondagattu), Jagtial Dist – 505501, T.S

(ACCREDITED BY NAAC A+ GRADE)

ABSTRACT

The proliferation of fake videos, fueled by advancements in deepfake technology, presents a significant challenge to the integrity of digital media. In this project, we propose a deep learning framework utilizing Long Short-Term Memory (LSTM) networks for the classification of fake videos. Our approach leverages the temporal dynamics inherent in video data to discern subtle differences between authentic and manipulated content.

We construct a dataset comprising both real and synthetic videos across various contexts and employ pre-trained LSTM models for feature extraction. Subsequently, these features are fed into a classification layer for binary decision-making.

Extensive experiments demonstrate the effectiveness of our method in accurately detecting fake videos, even in the presence of sophisticated manipulations. Furthermore, we explore the generalization of our model to unseen datasets and discuss its potential applications in combating the spread of misinformation and ensuring the authenticity of multimedia content.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER-1

INTRODUCTION

The advent of deep learning and artificial intelligence has brought both remarkable advancements and new challenges. One such challenge is the proliferation of fake videos, often referred to as deepfakes, which leverage sophisticated techniques to manipulate visual content convincingly. These manipulated videos pose serious threats to various sectors, including journalism, politics, and security.

In response to this growing concern, researchers and developers have been exploring methodsto detect and mitigate the spread of deepfake videos. One promising approach is the use of Long Short-Term Memory (LSTM) neural networks, a type of recurrent neural network (RNN)known for its ability to model sequential data effectively.

In this document, we present a project focused on the classification of fake videos using LSTM neural networks. By training the model on a dataset of both real and synthetic videos, we aim to develop a robust system capable of identifying deepfake content with high accuracy. Our project not only contributes to the ongoing efforts in combating misinformation but also showcases the potential of deep learning in addressing complex societal challenges.

1.1 Problem Description:

In the digital age, the prevalence of manipulated video content significantly threatens the authenticity of information, potentially leading to widespread misinformation. This project addresses this critical challenge by developing a robust machine learning model designedto accurately classify videos as real or fake. Utilizing Long Short-Term Memory (LSTM)networks and sophisticated image processing techniques, the system aims to enhance digital trust and reduce the impact of fake videos across various media platforms.

1.2 Objective:

The objective of this project is to develop a deep learning model based on Long Short-Term Memory (LSTM) neural networks to accurately classify videos as either authentic or manipulated (fake). The goal is to create a robust system capable of detecting various forms of video manipulation, including deepfakes, video slicing, and other forms of digital tampering. By training the model on a diverse dataset of both authentic and fake videos, weaim to achieve high accuracy and generalization performance, enabling the model to effectively discern subtle cues and anomalies indicative of video manipulation. This classification system will serve as a crucial tool in combating the spread of misinformationand protecting the integrity of visual media in various domains, including journalism, socialmedia, and security.

1.3 Existing System:

The current paradigm of fake video classification primarily relies on Convolutional Neural Networks (CNNs), recognized for their proficiency in image-based tasks. Existing systems leverage the spatial feature extraction capabilities of CNNs, forming the backbone of fake video methodologies. While these systems exhibit commendable success in discerning manipulated content within images, they encounter challenges in the temporal analysis of videos. The frame-level analysis characteristic of CNNs may lead to difficulties in capturing subtle temporal dynamics, contributing to occasional false positives and false negatives. As the landscape of fake video technology continues to evolve with more sophisticated techniques, there exists discernible need for advanced detection systems that can adapt to the nuanced temporalintricacies inherent in synthetic media. The effectiveness of this model is not that advantageousin this project and has the following limitations:

Disadvantages:

- Limited Temporal Understanding
- Large Computational Requirements
- Vulnerability to Adversarial Attacks
- Training data Imbalances

1.4 Proposed System:

The Long Short-Term Memory (LSTM) network is a specialized form of Recurrent Neural Network (RNN) designed to address the shortcomings of traditional RNNs, particularly in handling long-term dependencies in sequence data. This is achieved through a sophisticated architecture that includes several gates and states, enabling the network to regulate the flowof information effectively. Here’s a deeper look into the key components and operations within an LSTM unit:

Key Components of LSTM:

1.Cell State: The cell state acts as the "memory" of the LSTM unit and carries relevant information throughout the sequence processing. It ensures that informationthat is useful long-term can travel unchanged if needed.

2.Hidden State: The hidden state contains information derived from previous inputfeatures and is used for predictions. It is also passed to the next time step and can beused in the output sequence.

Gates in LSTM: Gates in LSTM control the flow of information using a sigmoid activation function, which outputs values between 0 (ignore this entirely) and 1 (let everything through).

- **.Forget Gate**: Decides what information to discard from the cell state. It looks at the previous hidden state (H_(t-1)) and the current input (x_t) and outputs a numberbetween 0 and 1 for each number in the cell state C_(t-1).
- **2.Input Gate and Candidate Layer:** The input gate decides which values willbe updated, and the candidate layer creates a vector of new candidate values that could be added to the state.

- **Output Gate**: The output gate decides what the next hidden state should be. Thehidden state contains information on previous inputs. The contents of the cell state arepassed through tanh (to push values to be between -1 and 1) and multiplied by the output of the output gate, so that only certain parts of the cell state are output.

3.Updating the Cell State: The old cell state, C_(t-1), is updated into the new cellstate C_t. The forget gate decides what is thrown away from the old cell state, and the input gate decides which new information is added from the candidate layer.

Implementation Details in the Proposed System:

In the proposed system, the LSTM processes the feature maps extracted from each frame ofthe video, leveraging the above mechanisms to analyze and remember essential temporal features necessary for detecting inconsistencies indicative of manipulated content. Each LSTM unit's operations, from gating to state updates, play a pivotal role in ensuring that both short-term detail.

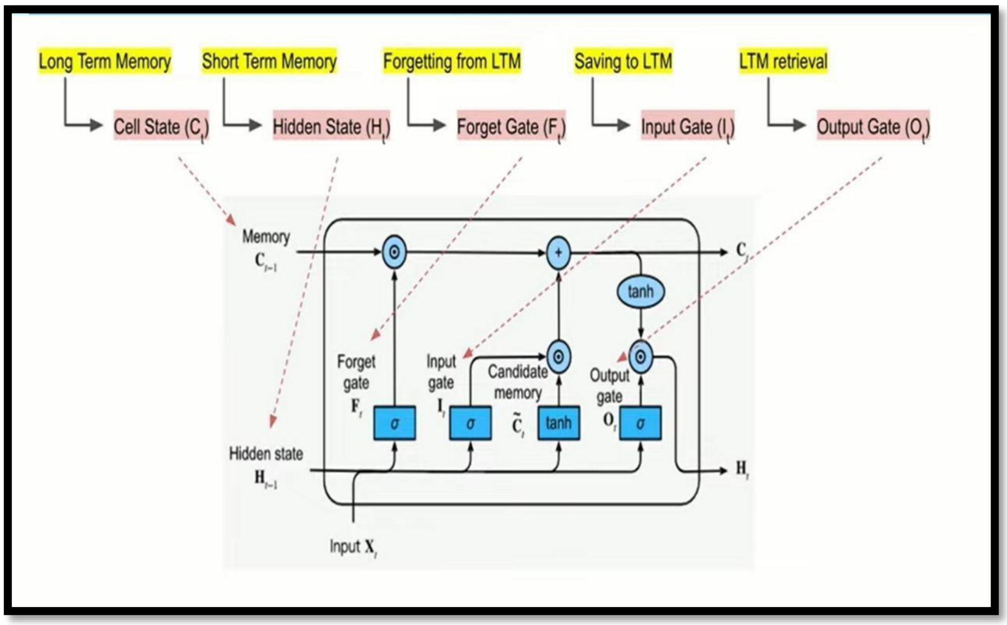


Fig no.1.4.1 LSTM Architecture Unit

CHAPTER-2
LITERATURE SURVEY

Face Warping Artifacts:

The approach to detect artifacts by comparing the generated face areas and their surrounding regionswith a dedicated Convolutional Neural Network model. 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 thefaces to be replaced in the source video. Their method has not considered the temporal analysis of the frames.

Detection by Eye Blinking:

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 thedeepfake generation algorithms have become so powerful that lack of eye blinking cannot be the onlyclue 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 eyebrowsetc.

Capsule networks to detect forged images and videos:

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. Ourmethod is proposed to be trained on noiseless and real time datasets.

Recurrent Neural Network (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 dataset consisting of just 600videos. 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 Realtimedata.

Synthetic Portrait Videos using Biological Signals:

Approach extract biological signals from facial regions on pristine and deepfake portrait video pairs.

CHAPTER-3

SYSTEM DESIGN

3.1 Hardware Requirements:

- Intel Xeon E5 2637- 3.5 GHz
- RAM-8 GB
- Hard Disk-100 GB
- Graphic card NVIDIA GeForce GTX Titan (6 GB)
- Operating System: Windows 7+

3.2 Software Requirements:

- Programming Language: Python
- IDE: Google Colab, Visual Studio Code
- Development Platforms: Anaconda Navigator
- Machine Learning Libraries: PyTorch, torchvision, OpenCV (cv2), NumPy, face_recognition, Keras
- Data Visualization Libraries: Matplotlib, Seaborn
- Algorithm: Long Short-Term Memory
- Web Framework: Flask (for building the website)

CHAPTER-4

PROPOSED ARCHITECTURE

4.1 Dataset Collection

To make the model efficient for real-time prediction, we gathered data from the Deepfake Detection Challenge (DFDC) dataset. We focused exclusively on this dataset to streamline the data collection process and ensure a balanced dataset for training.
We collected a total of 600 videos from the DFDC dataset, comprising 300 real and 300 fake videos. This balanced approach, with an equal number of real and fake videos, helps to avoid training bias and ensures accurate and real-time detection on various types of videos.

4.2 Data Preprocessing

In this step, the videos are preprocessed and all the unrequired and noise is removed from videos. Only the required portion of the video 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.

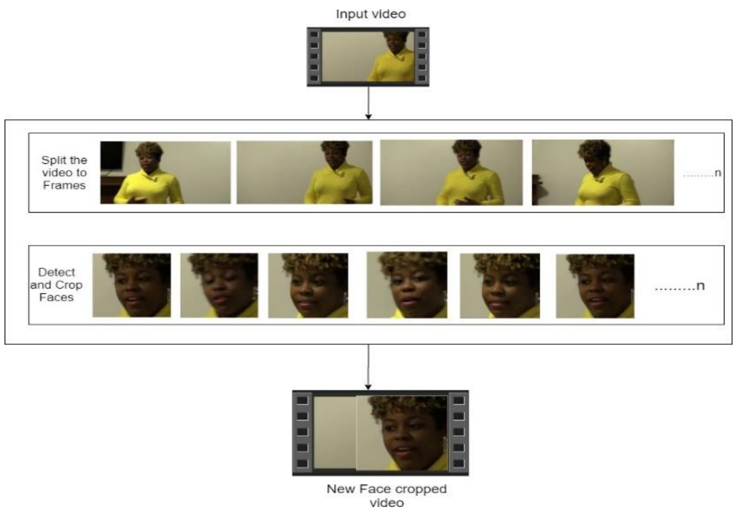


Fig no.4.2.1. Preprocessed video

4.3 Training the Model

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 fake 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.

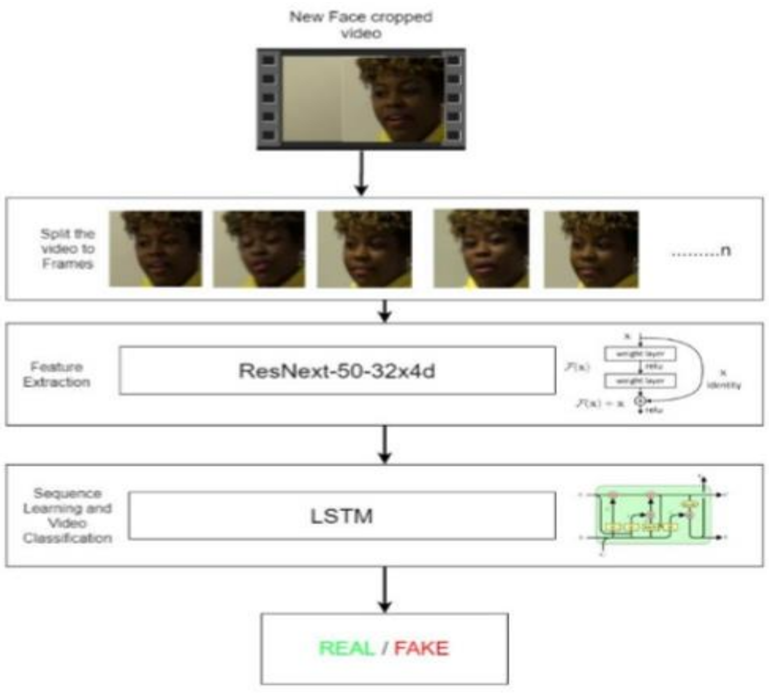


Fig no.4.3.1 Overview of the model

4.4 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.

4.5Development of Web Interface using Flask

Developing a user interface using Flask that enables users to interact with the fake video classification system. This interface allows users to upload videos and classifies them as real or fake, displaying the confidence percentage of the classification. Additionally, it provides a platform for users to give feedback, contributing to the ongoing refinement of the model.

4.6Creating and Installation of dependencies in Anaconda Navigator

The below dependencies should be installed in Anaconda Navigator for this project by creating an Anaconda environment.

```
pip install tensorflow
conda install -c conda-forge keras
conda install -c anaconda scikit-learn
conda install -c conda-forge opencv
conda install -c anaconda scikit-image
conda install -c anaconda flask
pip install pandas
pip install werkzeug==2.3.7
conda install pytorch torchvision torchaudio cudatoolkit=10.2 -c pytorch
conda install matplotlib
pip install face_recognition
```

CHAPTER-5

CODE IMPLEMENTATION

1. At first we should be developing and training the LSTM-based machine learning model which is essential for classifying videos as real or fake. This portion of the implementation involves preparing the dataset, selecting appropriate features through meticulous preprocessing, and then employing these features to train the LSTM model. The training process is carefully designed to optimize the model’s accuracy by fine-tuning various hyperparameters and employing techniques such as dropout to prevent overfitting. This code ultimately produces a robust trained model capable of detecting subtle manipulations in video content, which is critical for maintaining the integrity and trustworthiness of digital media.

2. Next part focuses on the implementation of a user-friendly web interface using Flask. This interface serves as the point of interaction for users, enabling them to upload videos and receive real-time classifications. The Flask framework was chosen for its simplicity and efficiency, allowing rapid deployment and easy scalability of the web application.

Together, these two codes encompass the complete system functionality from model training to user interaction, ensuring a seamless operation and user experience.

5.1 Code-1: Deep Learning Model Development

Mount the Drive in Google Colab and Import All Libraries

```
from google.colab import drive
drive.mount('/content/drive')

!pip3 install face_recognition

import glob
import numpy as np
import cv2
import torch
import torchvision
from torchvision import transforms
from torch.utils.data import DataLoader
from torch.utils.data import Dataset
import os
import matplotlib.pyplot as plt
import face_recognition
import pandas as pd
import seaborn as sn
from torch import nn
from torchvision import models
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from torch.optim import Adam
from torch.autograd import Variable
import sys
import time
import random
from tqdm.autonotebook import tqdm
```

Uploading the dataset:

```
# Directory containing the videos
directory = '/content/drive/MyDrive/raw_videos/'
files = glob.glob(directory + '*.mp4')
video_count = len(files)
print("Total number of videos:", video_count)
```

We collected a total of 646 videos from the DFDC dataset, comprising 323 real and 323 fake videos. This balanced approach, with an equal number of real and fake videos, helps to avoid training bias and ensures accurate and real-time detection on various types of videos.

1 . Data Preprocessing:

A)Splitting the frames

```
video_files = glob.glob('/content/drive/MyDrive/raw_videos/*.mp4')
frame_count = []
valid_video_files = []
for video_file in video_files:
 cap = cv2.VideoCapture(video_file)
    if int(cap.get(cv2.CAP_PROP_FRAME_COUNT)) >= 150:
        frame_count.append(int(cap.get(cv2.CAP_PROP_FRAME_COUNT)))
        valid_video_files.append(video_file)
    else:
        print(f"Skipping video: {video_file}, frame count less than 150")
print("Total number of videos:", len(valid_video_files))
print("Frames:", frame_count)
print("Total number of videos_frames:", len(frame_count))
print('Average frame per video:', np.mean(frame_count))
#to extract frames
def frame_extract(path):
  vidObj = cv2.VideoCapture(path)
  success = 1
  while success:
      success, image = vidObj.read()
      if success:
          yield image
#process the frames
!mkdir '/content/drive/MyDrive/preprocessed_before'
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 <= 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
    out.release()
```

B) Crop and Save Extracted Video Frames to a New Directory

create_face_videos(video_files,'/content/drive/MyDrive/preprocessed_before/')

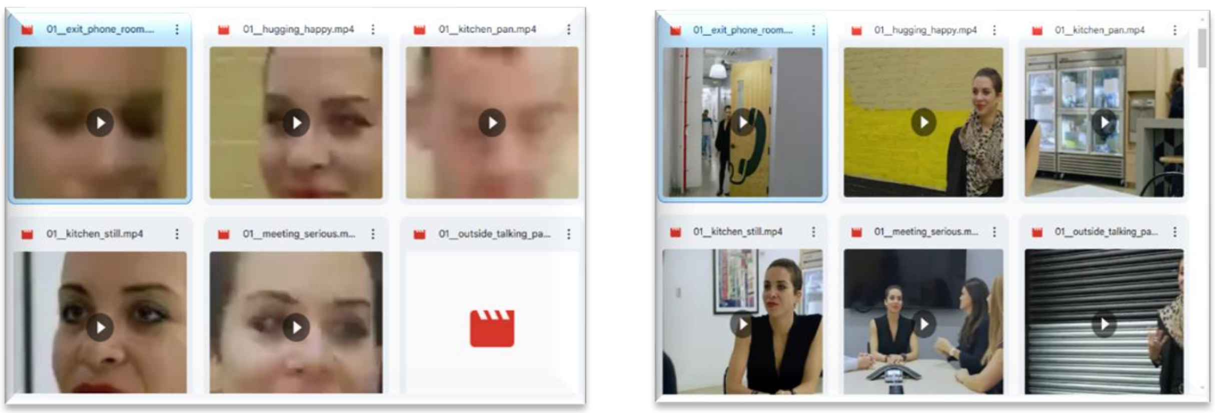


Fig no.5.1.1 Preprocessed outputs

- Using glob we imported all the videos in the directory in a python list.
- cv2.VideoCapture is used to read the videos and get the mean number of frames in each video.
- To maintain uniformity, based on mean a value 150 is selected as idea value for creating the new dataset.
- The video is split into frames and the frames are cropped on face location.
- The face cropped frames are again written to new video using VideoWriter.
- The new video is written at 30 frames per second and with the resolution of 112 x 112 pixels in the mp4 format.
- Instead of selecting the random videos, to make the proper use of LSTM for temporal sequence analysis the first 150 frames are written to the new video.

2.Training the data:

```
#THis code is to check if the video is corrupted or not, if the video is corrupted delete the
video.
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 frame 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/MyDrive/preprocessed_after/*.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))
#to load preprocessod video to memory
video_files =  glob.glob('/content/drive/My Drive/preprocessed_after/*.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
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]
        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]
        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/MyDrive/data1.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
```

This code is designed to preprocess video data by validating and filtering corrupted videos, extracting frames, and preparing datasets for training a deep learning model. It includes functions to check if a video is corrupted and remove it, extract frames from videos, and apply transformations to prepare the data. The script also calculates the number of frames per video, counts the real and fake videos, and organizes the data for model training. The **video_dataset** class helps in loading videos and their corresponding labels from a CSV file, ensuring the dataset is balanced and ready for training.

Splitting the data into TRAINING and TESTING

```
from sklearn.model_selection import train_test_split
header_list = ["file","label"]
labels = pd.read_csv('/content/drive/My Drive/data1.csv',names=header_list)
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))
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)
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,:,:,:])
```

Training the Model using LSTM Algorithm

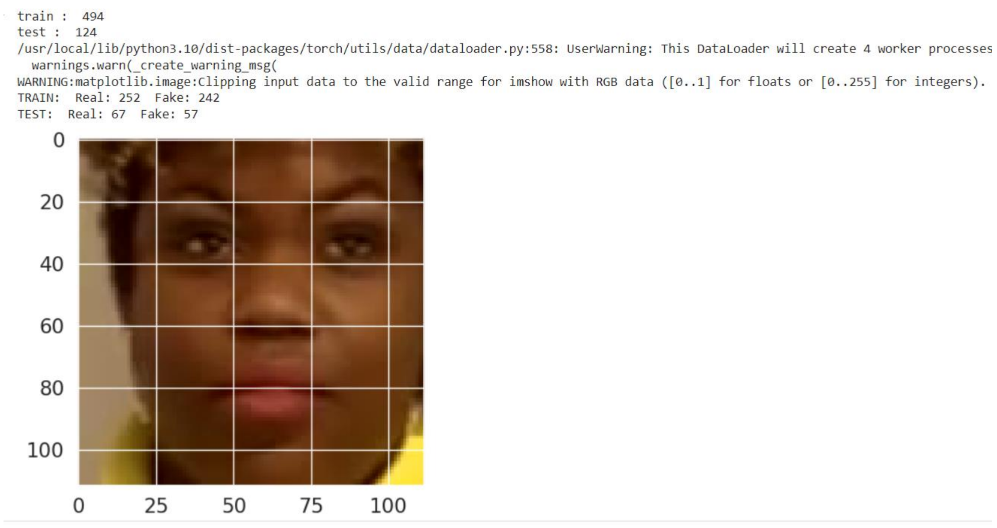
```
from torch import nn
from torchvision import models
class Model(nn.Module):
  def __init__(self, num_classes,latent_dim= 2048, lstm_layers=1 , hidden_dim = 2048,
bidirectional =    False):
      super(Model, self).__init__()
      model = models.resnext50_32x4d(pretrained = True)
      self.model = 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)))
```

Plotting Training and Validation Loss, Training and Validation Accuracy

```
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
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/drive/MyDrive/trained_model6.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
import seaborn as sn
#Output confusion matrix
def print_confusion_matrix(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    print('True positive = ', cm[0][0])
  print('False positive = ', cm[0][1])
    print('False negative = ', cm[1][0])
    print('True negative = ', cm[1][1])
    print('\n')
    df_cm = pd.DataFrame(cm, range(2), range(2))
    sn.set(font_scale=1.4)
    sn.heatmap(df_cm, annot=True, annot_kws={"size": 16})
    plt.ylabel('Actual label', size = 20)
    plt.xlabel('Predicted label', size = 20)
    plt.xticks(np.arange(2), ['Fake', 'Real'], size = 16)
    plt.yticks(np.arange(2), ['Fake', 'Real'], size = 16)
    plt.ylim([2, 0])
    plt.show()
    calculated_acc = (cm[0][0]+cm[1][1])/(cm[0][0]+cm[0][1]+cm[1][0]+ cm[1][1])
    print("Calculated Accuracy",calculated_acc*100)
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 #0.00001
num_epochs = 20
optimizer = torch.optim.Adam(model.parameters(), lr= lr,weight_decay = 1e-5)
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)



```
train :  494
test :  124
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:558: UserWarning: This DataLoader will create 4 worker processes
  warnings.warn(_create_warning_msg(
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
TRAIN:  Real: 252  Fake: 242
TEST:  Real: 67  Fake: 57
```

Fig no.5.1.2 Train_test_split output

Model Details

The model consists of following layers:

- **ResNext CNN:** The pre-trained model of Residual Convolution Neural Net work is used. The model name is resnext50_32x4d()[22]. This model consists of 50 layers and 32 x 4 dimensions.
- **Sequential Layer:** Sequential is a container of Modules that can be stacked together and run at the same time. Sequential layer is used to store feature vector returned by the ResNext model in a ordered way. So that it can be passed to the LSTM sequentially.
- **LSTM Layer:** LSTM is used for sequence processing and spot the temporal change between the frames.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.
- **ReLU:** A Rectified Linear Unit is activation function that has output 0 if the input is less than 0, and raw output otherwise. That is, if the input is greater than 0, the output is equal to the input. The operation of ReLU is closer to the way our biological neurons work. ReLU is non-linear and has the advantage of not having any backpropagation errors unlike the sigmoid function, also for larger Neural Networks, the speed of building models based off on ReLU is very fast.
- **Dropout Layer:** Dropout layer with the value of 0.4 is used to avoid over f itting in the model and it can help a model generalize by randomly setting the output for a given neuron to 0. In setting the output to 0, the cost function becomes more

sensitive to neighbouring neurons changing the way the weights will be updated during the process of backpropagation.

- **Adaptive Average Pooling Layer:** It is used To reduce variance, reduce computation complexity and extract low level features from neighbourhood.2 dimensional Adaptive Average Pooling Layer is used in the model.

Model Training Details

- **Train Test Split:** The dataset is split into train and test dataset with a ratio of 70%train videos and 30% 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.
- **Data Loader:** It is used to load the videos and their labels with a batch size of 4.
- **Training:** The training is done for 20 epochs with a learning rate of 1e-5(0.00001),weight decay of 1e-3 (0.001) using the Adam optimizer.
- **Adam optimizer[21]:** To enable the adaptive learning rate Adam optimizer with the model parameters is used.
- **Cross Entropy:** To calculate the loss function Cross Entropy approach is used because we are training a classification problem.
- **Softmax Layer:** A Softmax function is a type of squashing function. Squashing functions limit the output of the function into the range 0 to 1. This allows the output to be interpreted directly as a probability. Similarly, softmax functions are multi-class sigmoids, meaning they are used in determining probability of multiple classes at once. Since the outputs of a softmax function can be interpreted as a probability (i.e.they must sum to 1), a softmax layer is typically the final layer used in neural network functions. It is important to note that a softmax layer must have the same number of nodes as the output later. In our case softmax layer has two output nodes i.e REAL or FAKE, also Softmax layer provide us the confidence(probability) of prediction.
- **Confusion Matrix:** A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. This is the key to the confusion matrix. The confusion matrix shows the ways in which your classification model is confused when it makes predictions. It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made. Confusion matrix is used to evaluate our model and calculate the accuracy.
- **Export Model:** After the model is trained, we have exported the model. So that it can be used for prediction on real time data.

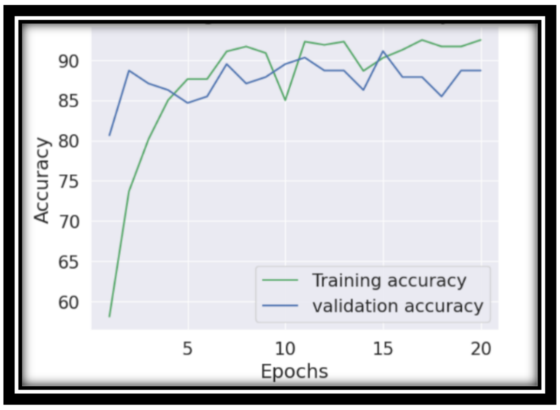


Fig no.5.1.3 Training and Validation Accuracy

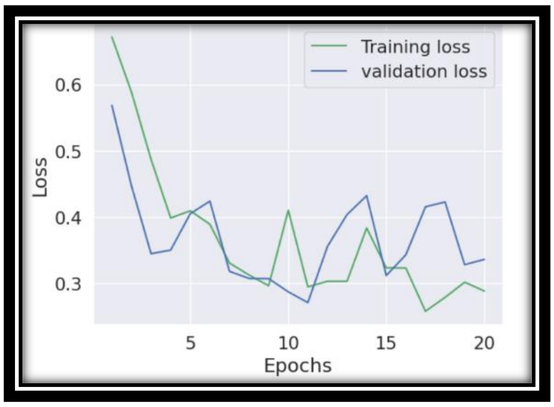
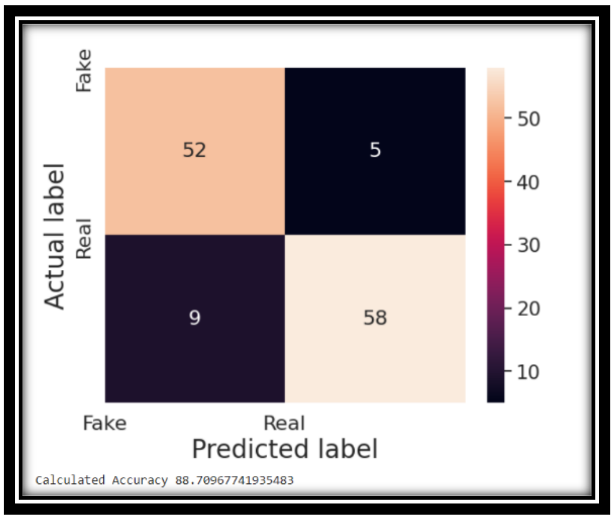


Fig no.5.1.4 Training and Validation loss

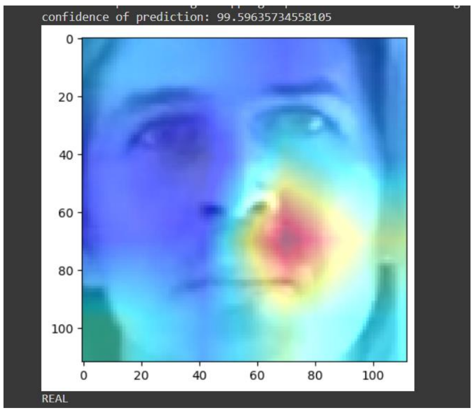


```
Calculated Accuracy 88.70967741935483
```

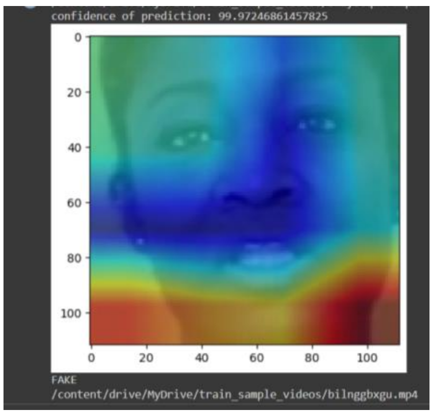
Fig no.5.1.5 Confusion Matrix

3. Prediction and Evaluation:

```
im_size = 112
mean=[0.485, 0.456, 0.406]
std=[0.229, 0.224, 0.225]
sm = nn.Softmax()
inv_normalize =  transforms.Normalize(mean=
1*np.divide(mean,std),std=np.divide([1,1,1],std))
def im_convert(tensor):
    """ Display a tensor as an image. """
    image = tensor.to("cpu").clone().detach()
    image = image.squeeze()
    image = inv_normalize(image)
    image = image.numpy()
    image = image.transpose(1,2,0)
    image = image.clip(0, 1)
    cv2.imwrite('./2.png',image*255)
    return image
def predict(model,img,path = './'):
  fmap,logits = model(img.to('cuda'))
  params = list(model.parameters())
  weight_softmax = model.linear1.weight.detach().cpu().numpy()
  logits = sm(logits)
  _,prediction = torch.max(logits,1)
  confidence = logits[:,int(prediction.item())].item()*100
  print('confidence of prediction:',logits[:,int(prediction.item())].item()*100)
  idx = np.argmax(logits.detach().cpu().numpy())
  bz, nc, h, w = fmap.shape
  out = np.dot(fmap[-1].detach().cpu().numpy().reshape((nc,
h*w)).T,weight_softmax[idx,:].T)
  predict = out.reshape(h,w)
  predict = predict - np.min(predict)
  predict_img = predict / np.max(predict)
  predict_img = np.uint8(255*predict_img)
  out = cv2.resize(predict_img, (im_size,im_size))
  heatmap = cv2.applyColorMap(out, cv2.COLORMAP_JET)
 img = im_convert(img[:,-1,:,:,:])
  result = heatmap * 0.5 + img*0.8*255
  cv2.imwrite('/content/1.png',result)
  result1 = heatmap * 0.5/255 + img*0.8
  r,g,b = cv2.split(result1)
  result1 = cv2.merge((r,g,b))
  plt.imshow(result1)
  plt.show()
class validation_dataset(Dataset):
    def __init__(self,video_names,sequence_length = 60,transform = None):
        self.video_names = video_names
        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)
        for i,frame in enumerate(self.frame_extract(video_path)):
            faces = face_recognition.face_locations(frame)
            try:
              top,right,bottom,left = faces[0]
              frame = frame[top:bottom,left:right,:]
            except:
              pass
            frames.append(self.transform(frame))
            if(len(frames) == self.count):
              break
        frames = frames[:self.count]
        return frames.unsqueeze(0)
    def frame_extract(self,path):
      vidObj = cv2.VideoCapture(path)
      success = 1
    while success:
        success, image = vidObj.read()
        if success:
            yield 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()
#Code for making prediction
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)])
path_to_videos= ["/content/drive/MyDrive/train_sample_videos/aelfnikyqj.mp4",
                 "/content/drive/MyDrive/train_sample_videos/bndybcqhfr.mp4",
                 "/content/drive/MyDrive/train_sample_videos/bilnggbxgu.mp4",
                 "/content/drive/MyDrive/train_sample_videos/bkmdzhfzfh.mp4"]
video_dataset = validation_dataset(path_to_videos,sequence_length = 10,transform =
train_transforms)
model = Model(2).cuda()
path_to_model = '/content/drive/MyDrive/trained_model6.pt'
model.load_state_dict(torch.load(path_to_model))
model.eval()
for i in range(0,len(path_to_videos)):
  print(path_to_videos[i])
  prediction = predict(model,video_dataset[i],'./')
  if prediction[0] == 1:
    print("REAL")
  else:
    print("FAKE")
```



```
confidence of prediction: 99.59635734558105
REAL
```



```
confidence of prediction: 99.97246861457825
FAKE
/content/drive/MyDrive/train_sample_videos/bilnggbxgu.mp4
```

Fig No.5.1.6 Predicted outputs

Model Prediction Details

- The model is loaded in the application
- The new video for prediction is preprocessed and passed to the loaded model for prediction
- The trained model performs the prediction and return if the video is a real or fake along with the confidence of the prediction

5.2 Code-2: Web Interface Using Flask

i)app.py

```
from flask import Flask, render_template, request, session
import torch
import torchvision
from torchvision import transforms
from torch.utils.data import DataLoader
from torch.utils.data.dataset import Dataset
from torchvision import models
from torch.autograd import Variable
import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
import face_recognition
import time
import sys
from torch import nn

app = Flask(__name__)

@app.route('/')
@app.route('/index')
def second():
   scrollValueText = 10
   return render_template('uploader.html', scrollValueText = scrollValueText)
app.secret_key = 'my_key'
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)
       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(x_lstm[:,-1,:]))
im_size = 112
mean=[0.485, 0.456, 0.406]
std=[0.229, 0.224, 0.225]

sm = nn.Softmax()
inv_normalize =  transforms.Normalize(mean=-1*np.divide(mean,std),std=np.divide([1,1,1],std))
def im_convert(tensor):
    """ Display a tensor as an image. """
    image = tensor.to("cpu").clone().detach()
    image = image.squeeze()
    image = inv_normalize(image)
    image = image.numpy()
    image = image.transpose(1,2,0)
    image = image.clip(0, 1)
    return image
def predict(model,img,path = './'):
  fmap,logits = model(img.to('cpu'))
  params = list(model.parameters())
  weight_softmax = model.linear1.weight.detach().cpu().numpy()
  logits = sm(logits)
  _,prediction = torch.max(logits,1)
  confidence = logits[:,int(prediction.item())].item()*100
  print('confidence of prediction:',logits[:,int(prediction.item())].item()*100)
  idx = np.argmax(logits.detach().cpu().numpy())
  bz, nc, h, w = fmap.shape
  out = np.dot(fmap[-1].detach().cpu().numpy().reshape((nc, h*w)).T,weight_softmax[idx,:].T)
  predict = out.reshape(h,w)
  predict = predict - np.min(predict)
  predict_img = predict / np.max(predict)
  predict_img = np.uint8(255*predict_img)
  out = cv2.resize(predict_img, (im_size,im_size))
  heatmap = cv2.applyColorMap(out, cv2.COLORMAP_JET)
  img = im_convert(img[:,-1,:,:,:])
  result = heatmap * 0.5 + img*0.8*255
  cv2.imwrite('/content/1.png',result)
  result1 = heatmap * 0.5/255 + img*0.8
  r,g,b = cv2.split(result1)
  result1 = cv2.merge((r,g,b))
  return [int(prediction.item()),confidence]
class validation_dataset(Dataset):
    def __init__(self,video_names, sequence_length, transform = None):
        self.video_names = video_names
        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)
        for i,frame in enumerate(self.frame_extract(video_path)):
            faces = face_recognition.face_locations(frame)
            try:
```

```
            top,right,bottom,left = faces[0]
            frame = frame[top:bottom,left:right,:]
          except:
            pass
          frames.append(self.transform(frame))
          if(len(frames) == self.count):
            break
        frames = torch.stack(frames)
        frames = frames[:self.count]
        return frames.unsqueeze(0)
    def frame_extract(self,path):
      vidObj = cv2.VideoCapture(path)
      success = 1
      while success:
          success, image = vidObj.read()
          if success:
              yield image
@app.route('/upload', methods=['POST'])
def upload():
    fileReader = request.files['file']
    scroll_value = int(request.form['scrollValue'])
    fileReader.save('./static/video/' + fileReader.filename)

    path_to_videos= ["./static/video/" + fileReader.filename]
    session['video_filename'] = fileReader.filename
    train_transforms = transforms.Compose([
                                        transforms.ToPILImage(),
                                        transforms.Resize((im_size,im_size)),
                                        transforms.ToTensor(),
                                        transforms.Normalize(mean,std)])
    pathProvider = path_to_videos[0]
    video_dataset = validation_dataset(path_to_videos,sequence_length = scroll_value,transform =
train_transforms)
    device = torch.device('cpu')
    model = Model(2).to(device)
    path_to_model = './models/trained_model6.pt'
    model.load_state_dict(torch.load(path_to_model, device))
    model.eval()
    predictions = ""
    for i in range(0,len(path_to_videos)):
        print(path_to_videos[i])
        prediction = predict(model,video_dataset[i],'./')
        accuracy = prediction[1]
        if prediction[0] == 1:
            prediction = "REAL"
        else:
            prediction = "FAKE"

    cap = cv2.VideoCapture(path_to_videos[0])
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    frame_interval = total_frames // int(scroll_value)

    frame_count = 0
    frame_index = 0
    frame_path = []
    face_index = 0
    face_path = []
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break
        if frame_count % frame_interval == 0:
            frame_path.append('./static/images/'+f'frame_{frame_index}.jpg')
            output_path = os.path.join('./static/images/', f'frame_{frame_index}.jpg')
            frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            face_locations = face_recognition.face_locations(frame_rgb)
            for (top, right, bottom, left) in face_locations:
                face_image = frame[top:bottom, left:right]
                face_path.append('./static/images/'+f'face_{face_index}.jpg')
                face_output_path = os.path.join('./static/images/', f'face_{face_index}.jpg')
                cv2.imwrite(face_output_path, face_image)
                face_index += 1
                if prediction == 'REAL':
                    cv2.rectangle(frame, (left, top), (right, bottom), (0, 255, 0), 2)
                else:
                    cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)
                label = f'{prediction}'
                font = cv2.FONT_HERSHEY_SIMPLEX
                font_scale = 1.5
                text_size = cv2.getTextSize(label, font, font_scale, 1)[0]
                text_left = left + 5
                text_top = top - text_size[1] - 5
                if prediction == 'REAL':
                    cv2.rectangle(frame, (text_left - 5, text_top - 5), (text_left + text_size[0] + 5, text_top
+ text_size[1] + 5), (0, 255, 0), cv2.FILLED)
                else:
                    cv2.rectangle(frame, (text_left - 5, text_top - 5), (text_left + text_size[0] + 5, text_top
+ text_size[1] + 5), (0, 0, 255), cv2.FILLED)
                cv2.putText(frame, label, (text_left, text_top + text_size[1]), font, font_scale, (0, 0, 0), 1,
cv2.LINE_AA)

            cv2.imwrite(output_path, frame)
            frame_index += 1
        frame_count += 1
    cap.release()
    return render_template('results.html',prediction=prediction, accuracy=accuracy,
frame_path=frame_path, video_path= '.'+pathProvider, face_path=face_path)

if __name__ == "__main__":
    app.run(debug=True)
```

ii.results.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Uploaded Video</title>
  <link rel="stylesheet" href="../static/css/uploadPage.css">
```

```
<script>
  window.onload = function () {
    const videoUrl = localStorage.getItem('uploadedVideo');
    if (videoUrl) {
      const videoElement = document.createElement('video');
      videoElement.src = videoUrl;
      videoElement.controls = true;
      const videoContainer = document.getElementById('video-container');
      videoContainer.appendChild(videoElement);
    } else {
      console.error('No uploaded video found.');
    }
  };
</script>
<style>
  body {
      overflow-y: scroll !important;
  }
  .container {
  background-repeat: repeat !important;
  background-size: contain;
}
  header {
        padding: 20px;
        color: #fff;
        display: flex;
        align-items: center;
      }

      header:hover{
          color: aquamarine;
      }

      header h1 {
        margin: 0;
        flex-grow: 1;
        text-align: center;
      }

      header a {
        color: white;
        text-decoration: none;
        font-size: 35px;
        font-weight: bold;
        margin-left: 20px;
        text-size-adjust: 20px;
      }

      header a:hover {
        color: #00ffea;
      }

      .first{
          position: absolute;
          top: 5%;
      }
      .second{
          position: absolute;
          top: 45%;
      }
      .third{
          position: absolute;
          top: 80%;
          left: 28%;
      }
      .first h1, .second h1, .third h1{
          color: white;
          position: relative;
          left: 50%;
      }
      .third p{
          position: relative;
          left: 10%;
      }
  #video-container {
    margin-left: 33%;
    max-width: 35%;
    max-height: 35%;
    border: 2px solid #000;
    display: flex;
    justify-content: center;
    align-items: center;
  }

  #video-container video {
    max-width: 100%;
    max-height: 100%;
  }
  .image-container,.face-container {
    width: 1250px;
    overflow-x: scroll;
    white-space: nowrap;
  }
  .image-container img {
    display: inline-block;
    height: auto;
    width: 300px;
    margin-right: 10px;
    margin-top: 30px;
  }
  .face-container img{
    display: inline-block;
    height: 150px;
    width: 150px;
    margin-right: 10px;
    margin-top: 30px;
  }
</style>
</head>
<body>
```

```
  <header>
    <a href="/upload">FAKE VIDEO DETECTION</a>
  </header>
  <div class="container" style="justify-content: center;">
    <div class="first">
        <h1>Extracted Frames</h1>
        <div class="image-container" style="margin-left: 10%">
        {% for frame in frame_path %}
        <img src= {{ frame }}>
        {% endfor %}
        </div>
    </div>
    <div class="second">
        <div class="face-container" style="margin-left: 10%">
            {% for frame in face_path %}
            <img src={{ frame }}>
            {% endfor %}
        </div>
    </div>
    <div class="third">
    <h1 style="margin-top:60px">Results</h1>
    <div style="margin-top: 20px;text-align: center;">
      <div style="text-align: center;">
        {% if prediction == 'REAL' %}
            <p style="font-size: 25px; color: rgb(57, 255, 20); font-weight: bold;">PREDICTION =
REAL</p>
        {% else %}
            <p style="font-size: 25px; color: red; font-weight: bold;">PREDICTION = FAKE</p>
        {% endif %}
    </div>
    <div style="margin-top: 20px; text-align: center;">
        <p style="font-size: 25px; color: {% if prediction == 'REAL' %}rgb(57, 255, 20){% else
%}red{% endif %}; font-weight: bold;">Confidence Percentage = {{ accuracy }}</p>
    </div>
  </div>
    </div>
</body>
</html>
```

iii.uploader.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Fake video Classification using LSTM</title>
    <link rel="stylesheet" href="../static/css/uploadPage.css">
    <script>
        window.onload = function() {
        const scrollBar = document.getElementById('scroll-bar');
```

```
  const scrollThumb = document.getElementById('scroll-bar-thumb');
  const scrollValue = document.getElementById('scroll-value');
  scrollBar.addEventListener('click', function(event) {
  const clickX = event.clientX - scrollBar.getBoundingClientRect().left;
  const scrollPercentage = clickX / scrollBar.offsetWidth;
  let scrollValueText;
  if (scrollPercentage < 0.15) {
    scrollValueText = 10;
  } else if (scrollPercentage < 0.35) {
    scrollValueText = 20;
  } else if (scrollPercentage < 0.55) {
    scrollValueText = 40;
  } else if (scrollPercentage < 0.75) {
    scrollValueText = 60;
  } else if (scrollPercentage < 0.95) {
    scrollValueText = 80;
  } else {
    scrollValueText = 100;
  }
  scrollThumb.style.width = scrollPercentage * 100 + '%';
  scrollValue.textContent = scrollValueText;
  const scrollValueInput = document.getElementById('scrollValueInput');
  scrollValueInput.value = scrollValueText;
});
};
  function handleFileSelect(event) {
    const file = event.target.files[0];
    const reader = new FileReader();
    console.log(" I am here ")
    reader.onload = function (e) {
      const videoUrl = e.target.result;
      localStorage.setItem('uploadedVideo', videoUrl);
      window.location.href = '/upload';
    };
    reader.readAsDataURL(file);
  }
</script>
<style>
  header {
    padding: 20px;
    color: #fff;
    display: flex;
    align-items: center;
  }
  header:hover{
      color: aquamarine;
  }
  header h1 {
    margin: 0;
    flex-grow: 1;
    text-align: center;
  }
  header a {
    color: white;
    text-decoration: none;
```

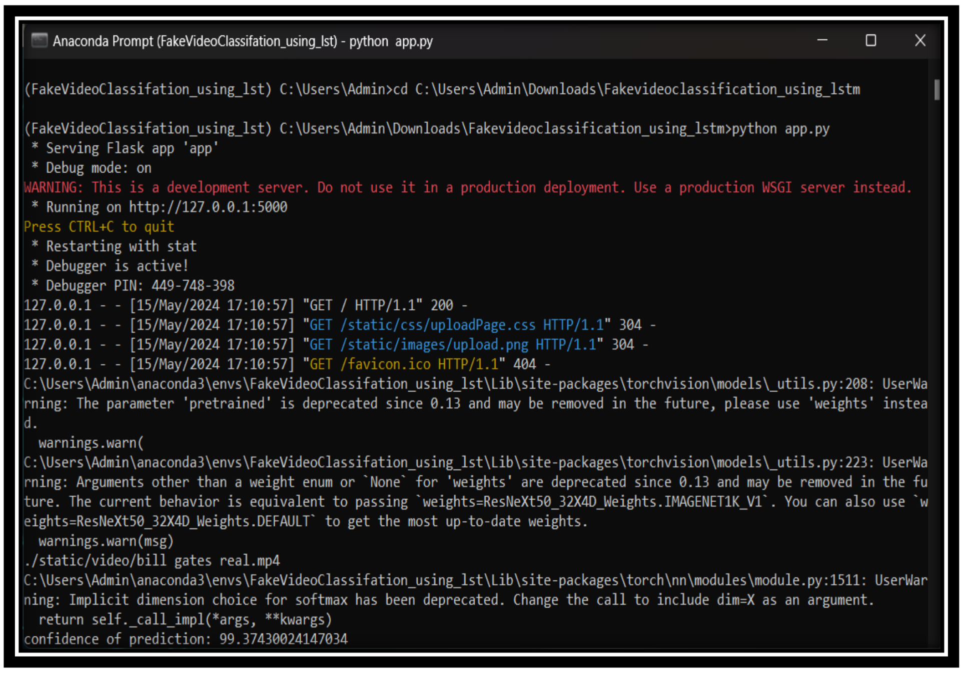
```
          font-size: 35px;
          font-weight: bold;
          margin-left: 20px;
          text-size-adjust: 20px;
        }
        header a:hover {
          color: #00ffea;
        }
        #upload-form {
            position: absolute;
            top: 15%;
            left: 35%;
          width: 300px;
          margin: 0 auto;
          margin-top: 50px;
          border: 2px solid #a9a7a7;
          padding: 50px;
          border-radius: 5px;
          box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
        }
        .no-background-image {
          background-color: transparent;
        }
        #upload-form input[type="file"] {
          margin-bottom: 10px;
          background-color: white;
        }
        #scroll-bar-container {
          width: 300px;
          margin: 50px auto;
          position: absolute;
          top: 50%;
          left: 38%;
        }
        #scroll-bar {
          width: 100%;
          height: 20px;
          background-color: #f1f1f1;
          position: relative;
          cursor: pointer;
        }
        #scroll-bar-thumb {
          width: 0;
          height: 100%;
          background-color: #007bff;
          position: absolute;
        }
        #scroll-value {
          margin-top: 10px;
          text-align: center;
          font-size: 18px;
          color: white;
        }
      </style>
  </head>
```

```
<body>
 <header>
   <a href="/upload">
     <span style="color: white; font-weight: bold; font-family: 'Times New Roman', Times, serif;
font-size: 50px;">
           <span id="content"></span>
       </span>
   </a>
</header>
<script>
   const text = "Fake video Classification using LSTM";
   const contentElement = document.getElementById("content");
   let index = 0;
   function displayText() {
       if (index < text.length) {
           contentElement.textContent += text.charAt(index);
           index++;
           setTimeout(displayText, 130);
       }
   }
   displayText();
</script>
   <div class="container" style="justify-content: center;">
   </div>
   <div id="upload-form" class="no-background-image">
       <h2 style="text-align: center;color: white;font-size: 30px;">Upload Video</h2>
       <form method="post" enctype="multipart/form-data" action="/upload">
         <input type="file" name="file">
         <input type="text" name="scrollValue" value="{{ scrollValueText }}" style="display:
none;" id="scrollValueInput">
         <input type="submit" value="Upload" class="getStarted">
     </form>
     </div>
     <div id="scroll-bar-container">
       <div id="scroll-bar">
         <div id="scroll-bar-track"></div>
         <div id="scroll-bar-thumb"></div>
         <div id="drag-handle"></div>
       </div>
       <div id="scroll-value">0</div>
     </div>
</body>
</html>
```

- The User Interface for the application is developed using Flask framework.
- The first page of the User interface uploader.html contains a tab to browse and upload the video. The uploaded video is then passed to the model and prediction is made by the model. The model returns the output whether the video is real or fake along with the confidence of the model. The output is rendered in the results.html of the face of the playing video.

CHATPER-6
OUTPUT SCREENS

6.1 OUTPUT SCREENS



```
Anaconda Prompt (FakeVideoClassifation_using_lst) - python  app.py

(FakeVideoClassifation_using_lst) C:\Users\Admin>cd C:\Users\Admin\Downloads\Fakevideoclassification_using_lstm

(FakeVideoClassifation_using_lst) C:\Users\Admin\Downloads\Fakevideoclassification_using_lstm>python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 449-748-398
127.0.0.1 - - [15/May/2024 17:10:57] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [15/May/2024 17:10:57] "GET /static/css/uploadPage.css HTTP/1.1" 304 -
127.0.0.1 - - [15/May/2024 17:10:57] "GET /static/images/upload.png HTTP/1.1" 304 -
127.0.0.1 - - [15/May/2024 17:10:57] "GET /favicon.ico HTTP/1.1" 404 -
C:\Users\Admin\anaconda3\envs\FakeVideoClassifation_using_lst\Lib\site-packages\torchvision\models\_utils.py:208: UserWa
rning: The parameter 'pretrained' is deprecated since 0.13 and may be removed in the future, please use 'weights' instea
d.
  warnings.warn(
C:\Users\Admin\anaconda3\envs\FakeVideoClassifation_using_lst\Lib\site-packages\torchvision\models\_utils.py:223: UserWa
rning: Arguments other than a weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed in the fu
ture. The current behavior is equivalent to passing `weights=ResNeXt50_32X4D_Weights.IMAGENET1K_V1`. You can also use `w
eights=ResNeXt50_32X4D_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
./static/video/bill gates real.mp4
C:\Users\Admin\anaconda3\envs\FakeVideoClassifation_using_lst\Lib\site-packages\torch\nn\modules\module.py:1511: UserWar
ning: Implicit dimension choice for softmax has been deprecated. Change the call to include dim=X as an argument.
  return self._call_impl(*args, **kwargs)
confidence of prediction: 99.37430024147034
```

Fig no.6.1.1 Output Screen

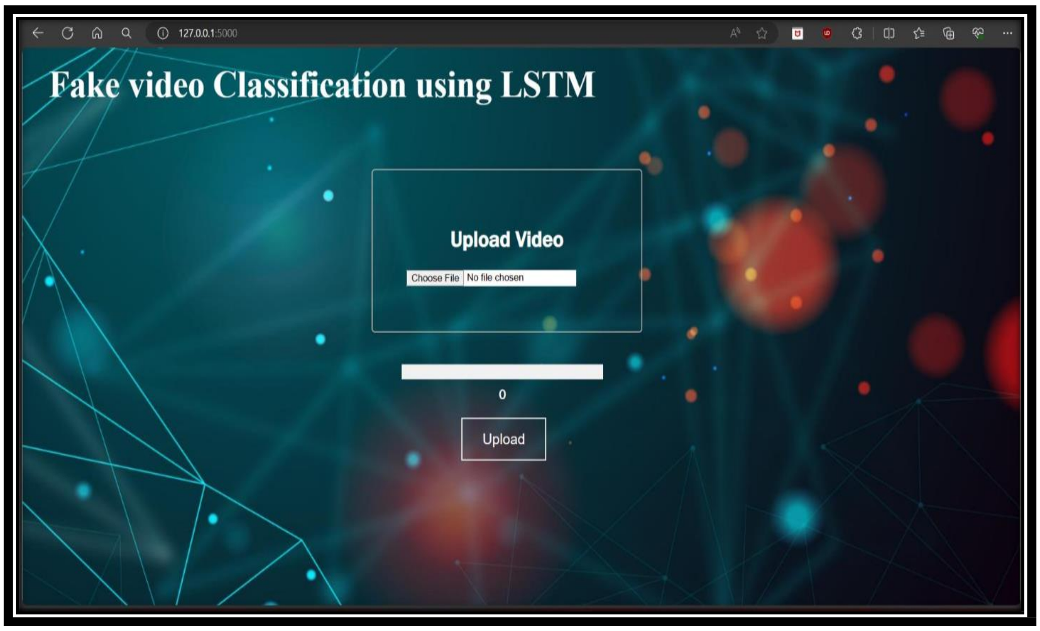
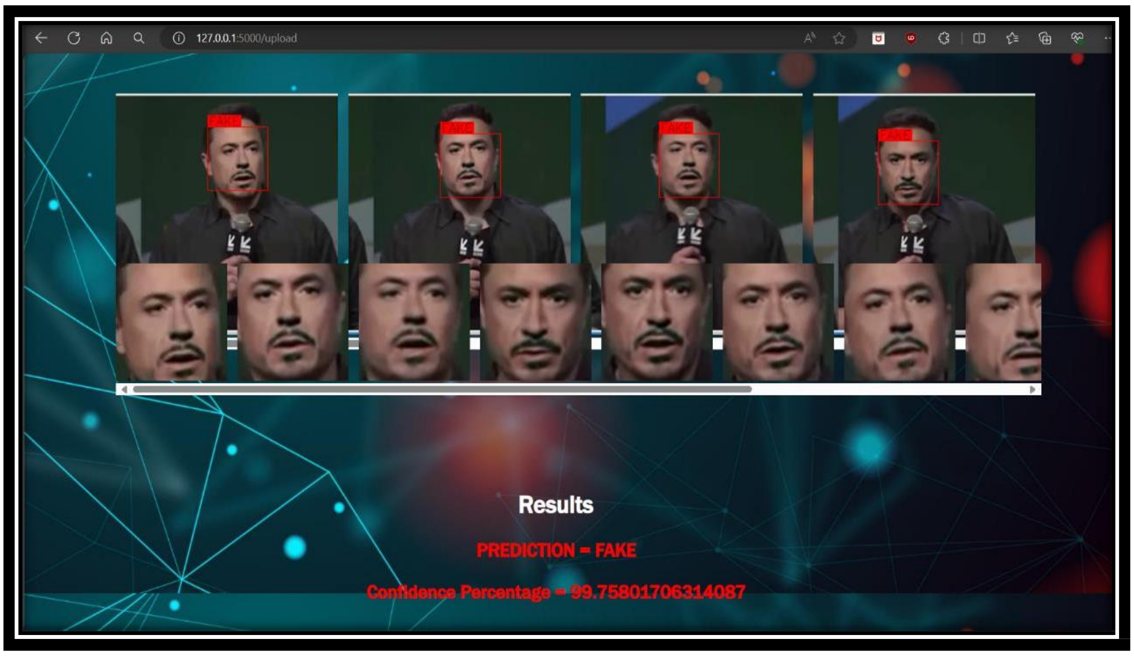


Fig no.6.1.2 Web Interface

1. When uploading the Fake video it gives the following output



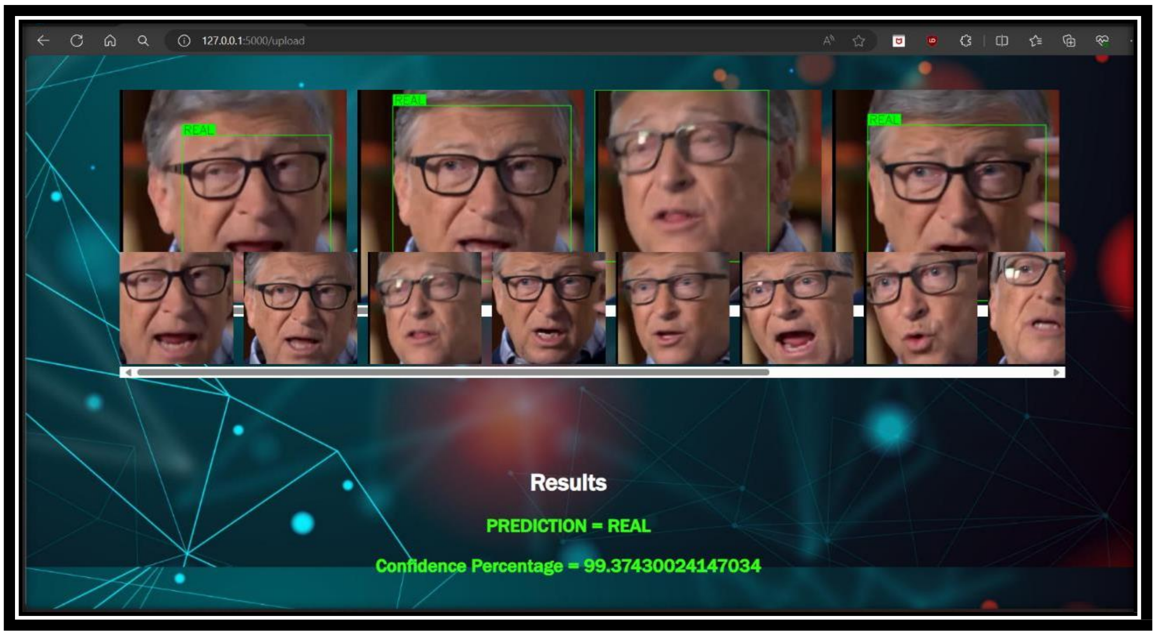
Results

PREDICTION = FAKE

Confidence Percentage = 99.75801706314087

Fig.no 6.1.3 Fake video output

2. When uploading the Real video it gives the following output



Results

PREDICTION = REAL

Confidence Percentage = 99.37430024147034

Fig.no 6.1.4 Real video output

CHAPTER-7

RESULTS AND DISCUSSIONS

Model Performance

In this section, we present the performance metrics of the fake video classification model, including the training and validation loss, training and validation accuracy, and the confusion matrix.

1. Training and Validation Loss:

Training and validation loss are calculated to evaluate how well the model is learning and generalizing. The loss is typically calculated using a loss function such as cross-entropy loss for classification problems.

- Cross-Entropy Loss: This loss measures the performance of a classification model whose output is a probability value between 0 and 1.

  $\textbf{Loss}=-N1\sum_{i=1}^{N} [yi \log(pi)+(1-yi) \log(1-pi)]$

where yi is the true label, pi is the predicted probability, and N is the number of samples.

2. Training and Validation Accuracy:

- Accuracy is a measure of the number of correct predictions made by the model divided by the total number of predictions. It is calculated as follows:

  Accuracy=Total Number of Predictions/Number of Correct Predictions

3. Confusion Matrix:

- The confusion matrix is a table used to describe the performance of a classification model on a set of test data for which the true values are known. It includes four components:

1. True Positives (TP): The number of correct positive predictions.
2. True Negatives (TN): The number of correct negative predictions.
3. False Positives (FP): The number of incorrect positive predictions.
4. False Negatives (FN): The number of incorrect negative predictions.

Using these components, several metrics can be derived:

From the fig 5.1.3, fig 5.1.4, fig 5.1.5 the values are:

True Positives (TP): 52

True Negatives (TN): 58

False Positives (FP): 5

False Negatives (FN): 9

Precision:

Precision=$TP/TP+FP$
Precision=52/52+5=52/57≈0.9123

Recall (Sensitivity):

Recall=$TP/TP+FN$

Recall=52/52+9=52/61≈0.8525

F1 Score: The harmonic mean of precision and recall.

F1 Score=2×(Precision×Recall / Precision+Recall)

F1 Score=2×(0.9123x0.8525 / 0.9123+0.8525)≈0.8817

Overall Accuracy:

Accuracy=$TP+TN$ / $TP+TN+FP+FN$

Accuracy=52+58 / 52+58+5+9=110/124≈0.8871

This results in an overall accuracy of approximately 88.71%.

So, the calculated values are:

- Precision: ≈0.9123
- Recall: ≈0.8525
- F1 Score: ≈0.8817
- Overall Accuracy: ≈0.8871

4. Confidence Percentage:

For each video classified as real or fake, the model also provides a confidence percentage indicating the certainty of the prediction. This confidence score helps in assessing the reliability of each prediction.

By presenting these results, we can see that the model performs well with an overall accuracy of approximately 88.71%. The training and validation metrics indicate that the model has learned effectively and generalizes well to new data. The confusion matrix provides a detailed view of the classification performance, highlighting the balance between correctly identified real and fake videos. Other performance metrics such as precision, recall, and F1 score further validate the effectiveness of the model.

CHAPTER 8
CONCLUSION

The culmination of the fake video classification project employing LSTM models offers a compelling narrative of progress in the ongoing battle against visual misinformation. Through meticulous experimentation and analysis, the project has unveiled the potential of LSTM architectures to discern between authentic and manipulated video content with notable accuracy. By harnessing the temporal dependencies ingrained within video sequences, the LSTM model exhibits a remarkable ability to detect nuanced alterations and fabrications, thereby enhancing its utility in combating the proliferation of fake videos across digital platforms. Despite encountering challenges such as dataset imbalance and varying degrees of manipulation complexity, the project adeptly navigated these obstacles through rigorous data preprocessing and model optimization, ultimately reinforcing the credibility of its findings. Methodologically, the decision to employ LSTM proves astute, leveraging its innate capacity to capture long-term dependencies within sequential data, a crucial asset in decoding the nuanced patterns present in fake videos. The project'smethodology underscores the significance of meticulous feature engineering, model architecture optimization, and robust evaluation protocols in achieving reliable and interpretable results. Looking ahead, the project not only opens avenues for further refinement and enhancementof LSTM-based approaches but also underscores the interdisciplinary significance of collaboration between machine learning experts, multimedia analysts, and information verification specialists in combating visual misinformation. In essence, the project represents a milestone in the quest to safeguard the integrity and credibility of visual content in an era characterized by the rapid dissemination of digital misinformation.

CHAPTER-9
REFERENCES

[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] Deepfake detection challenge dataset : https://www.kaggle.com/c/deepfake-detection challenge/data

[3] 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

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

[5] Yuezun Li, Siwei Lyu, “ExposingDF Videos By Detecting Face Warping Arti facts,” sin arXiv:1811.00656v3.

[6] TensorFlow: https://www.tensorflow.org/

[7] Keras: https://keras.io/

[8] PyTorch : https://pytorch.org/

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

[10] 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.

[11] Face app: https://www.faceapp.com/

[12] Face Swap : https://faceswaponline.com/

[13] Yuezun Li, Ming-Ching Chang and Siwei Lyu “Exposing AI Created Fake Videos by Detecting Eye Blinking” in arXiv:1806.02877v2.

[14] Huy H. Nguyen , Junichi Yamagishi, and Isao Echizen “ Using capsule net works to detect forged images and videos ” in arXiv:1810.11215.