DEEPFAKE FACE DETECTION

Dissertation submitted

in partial fulfillment of requirements for the award of the degree of

Master of Computer Applications (MCA)

Submitted By

MATSA MURALI KRISHNA

(Reg No: 322203320038)

Under the esteemed guidance of

Dr. N. Jaya Lakshmi

Associate Professor

Department of Computer Applications

COLLEGE OF ENGINEERING
(AUTONOMOUS)

Department of Computer Applications

GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING

(AUTONOMOUS)

(Affiliated to Andhra University, A.P)

VISAKHAPATNAM – 530048

2022-2024

CERTIFICATE

GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING

(AUTONOMOUS)

This is to certify that, the dissertation titled **“DEEPFAKE FACE DETECTION”** is submitted by **Mr. MATSA MURALI KRISHNA** with Regd. No **322203320038** in partial fulfillment of the requirement for the award of the Degree of **M.C.A** in **GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING (AUTONOMUS)** affiliated to Andhra University, Visakhapatnam is a bonafide record of project carried out by him under my guidance and supervision.

The contents of the project report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree.

Internal Guide **Head of the Department**

External Examiner

CERTIFICATE OF PLAGIARISM CHECK

COLLEGE OF ENGINEERING
(AUTONOMOUS)

This is to certify the Master of Computer Applications (MCA) dissertation submitted by MATSA MURALI KRISHNA (322203320038) of the Department of Computer Applications underthe supervision of Dr. N. Jaya Lakshmi Associate Professor of Department of Computer Applications has undergone plagiarism check and found to have similarity index less than 40%. The details of the plagiarism check are as under.

File Name	: **Deepfake Face Detection.pdf**
Dissertation Title	: **Deepfake Face Detection**
Date and Time of Submission	:
Submission ID	:
Similarity Index	:

Dean- Academics Programs (PG)

DECLARATION

I hereby declare that the dissertation titled **"DEEPFAKE FACE DETECTION"** is submitted to the Department of Computer Applications, **Gayatri Vidya Parishad College of Engineering (Autonomous)** affiliated to Andhra University, Visakhapatnam in partial fulfillment of the requirements for the award of the Degree of **Master of Computer Applications (M.C.A).** This work is done by me and authentic to the best of my knowledge under the direction and valuable guidance of **Dr. N.Jaya Lakshmi, M.Tech, Ph.D,** Associate Professor.

(MATSA MURALI KRISHNA)

(Regd. No. 322203320038)

ACKNOWLEDGEMENT

I take the opportunity to thank everyone who has contributed to making the project possible. I am thankful to **Gayatri Vidya Parishad College of Engineering (Autonomous)** for giving opportunity me to work on a project as part of the curriculum.

I offer my copious thanks to **Dr. A. BalaKoteswara Rao**, the Principal, **Gayatri Vidya Parishad College of Engineering (Autonomous),** for providing the best faculty and lab facilities throughout the completion of Master of Computer Applications course.

I offer my copious thanks to Prof. **Dr. K. Narasimha Rao** Professor & Dean, Academics (PG), for his valuable suggestions and constant motivation that greatly helped me to complete project successfully.

I offer my copious thanks to **Dr. Y. Anuradha**, Associate Professor & Head, Department of Computer Applications, for her valuable suggestions and constant motivation that greatly helped me to complete the project successfully.

My sincere thanks to **Dr. N. Jaya Lakshmi**, Associate Professor, and my project guide, for her constant support, encouragement, and guidance. I am very much grateful for her valuable suggestions.

(MATSA MURALI KRISHNA)

(Reg. No 322203320038)

ABSTRACT

Deepfake videos, created using advanced AI and machine learning, pose a significant threat across various fields like politics, entertainment, and cybersecurity. They're highly realistic and can spread false information, sparking public concern. To address this, our project proposes a robust deepfake detection system. By combining Inception V3 and GRU algorithms, we train the system to identify key video features and detect manipulations effectively. Inception V3 extracts frame-level features, which are then used to train a GRU-based model. This approach achieves an 80% accuracy rate in distinguishing between real and manipulated videos.

INDEX

Certificate…………………………………………………………….ii

Certificate for Plagiarism check………………………………...iii

Declaration……………………………………………………....iv

Acknowledgement………………………………………………v

Abstract…………………………………………………………vi

Index……………………………………………………………vii

Contents………………………………………………………...viii

List of Figures…………………………………………………....x

List of Tables…………………………………………………….xi

CONTENTS

CHAPTER NAME **PAGE NO**

1. INTRODUCTION…... 1-2

2. LITERATURE SURVEY.. 3-4

3. SYSTEM ANALYSIS……………………………………………………………….5-6

3.1 Existing System……………………………………………………………….6

3.2 Proposed System……………………………………………………………….6

4. SOFTWARE REQUIREMENTS AND SPECIFICATION…………………………7-10

4.1 Hardware Requirements…………………………………………………8

4.2 Software Requirements…………………………………………………8

4.3 Libraries Used……………………………………………………………….8-10

a. Flask……………………………………………………………..8

b. Tensor Flow…………………………………………………………8

c. Keras.. 9

d. Numpy..9

e. Pandas 9

f. CV2…..9

5. SYTEM DESIGN……………………………………………………………………..10

5.1 System Implementation………………………………………………….……11

5.2 UML Diagrams……………………………………………………………12-13

5.3 Flow Chart……………………………………………………………………14

6. IMPLEMENTATION ..…………………………………………………………..15-38

6.1 Algorithm.. 16

6.2 Code…………………………………………………………………………...17-38

7. OUTPUT SCREENS…………………………………………………………..39-42

8. SYSTEM TESTING…………………………………………………………...43-45

9. CONCLUSION AND FUTURE SCOPE…………………………………….46-47

REFERENCES…………………………………………………………………...48

LIST OF FIGURES

FIGURE NO	NAME OF THE FIGURE	PAGE NO
5.1	SYTEM IMPLEMENTATION	12
5.2.1	USER USECASE DIAGRAM	13
5.2.2	ADMIN USE CASE DIAGRAM	13
5.3	DEEPFAKE DETECTION PROCESS	14
7.1	WEBPAGE PREVIEW	40
7.1.1	VIDEO UPLOADED PAGE	41
7.1.2	RESULT PAGE	42

LIST OF TABLES

TABLE NO	TABLE NAME	PAGE NO
8.1	TEST CASE	45

CHAPTER 1

INTRODUCTION

INTRODUCTION

Deepfakes, realistic videos made by swapping faces, are a big worry in today's social media world. They can cause political trouble, make fake terror events, or share revenge porn, easily tricking people. For instance, there are deepfake videos showing Brad Pitt and Angelina Jolie in compromising situations.

It's really important to tell if a video is fake or real. We're using AI to fight against fake videos made by AI. Deepfakes are created using tools like Face App and Face Swap, which use fancy computer programs. Our way of spotting them uses a special kind of AI called a neural network. It looks at each frame of the video and compares it to real videos to see if anything looks weird. We trained it by showing it lots of fake and real videos so it can learn the difference. We used big collections of videos from different sources to teach it to recognize fakes better.

To make it easy for customers, we made an app where you can upload a video. Our system checks if it's real or a deepfake, then shows you the result and how confident it is. Nowadays, phones have really good cameras and it's easy to share videos online. Thanks to advanced tech like deep learning, we can do amazing things like make super realistic videos and even generate music or speech. These technologies help in many ways, like making text-to-speech tools and creating images for medical research.

CHAPTER 2

LITERATURE SURVEY

LITERATURE SURVEY

1. **Peipeng Yu and Zhihua Xia,** reviewed the current state of research on detecting deepfake videos. They found that existing detection methods aren't good enough for real-world use. They suggest that future research should focus on making detection methods more general and robust.

2. **G¨uera and Delp,** looked into using neural networks to detect deepfakes by analyzing facial features frame by frame. They propose that adding more layers to the networks could improve the quality of the detection.

3. **Korshunov and Marcel,** found that deepfake videos created using GANs pose challenges for both face recognition systems and current detection methods. They suggest that algorithms focusing on measuring visual consistency perform better for high-quality deepfakes compared to those focusing on inconsistency in the video.

4. **Yang et al**, and colleagues discuss how advancements in technology have made generating fake images, particularly deepfakes, more accurate and believable. They highlight that the main challenge lies in improving the quality of the generated images, suggesting that training confrontational deepfake models could degrade the quality of synthesized images.

CHAPTER 3

SYSTEM ANALYSIS

SYSTEM ANALYSIS

3.1 Existing System:

Current systems used to detect deepfakes may rely on traditional methods like rule-based systems, basic video analysis, and face recognition. However, these methods might not be advanced enough to accurately detect the increasingly realistic deepfake videos created using sophisticated AI and machine learning techniques.

3.2 Proposed System:

The suggested system, which combines Inception V3 and GRU, marks a step forward from traditional approaches. Inception V3 is renowned for its capacity to extract significant features from images, while GRU excels at capturing sequential dependencies, making them a fitting pair for analyzing videos.

CHAPTER 4

SOFTWARE REQUIREMENTS AND SPECIFICATION

- **4.1 HARDWARE REQUIREMENTS:**
- Processor: Intel i3v or higher
- RAM: 8 GB or higher
- Hard Disk : 50 GB

- **4.2 SOFTWARE REQUIREMENTS:**
- Operating System : Windows 7/8/10/11, MacOS
- IDE: Visual Studio
- Technology Used: HTML, CSS, Java script , Python, Flask

- **LIBRARIES USED:**

- **A.Flask**: Flask is a lightweight web application for Python. It is designed to be fast and simple to begin with, yet capable of scaling up to handle complex applications. Flask offers developers a variety of tools and libraries to assist in the development of web applications.
- **B.Tensor Flow:** TensorFlow has emerged as a cornerstone in deep learning projects, offering a robust framework for building and deploying machine learning models. Its extensive collection of pre-built modules and libraries streamlines common tasks such as data preprocessing, model training, and evaluation, expediting the development cycle.
  - **C.Keras:** Keras stands out as a high-level neural networks API, widely embraced for its simplicity, flexibility, and user-friendly interface in deep learning projects. Its extensive library of pre-trained models and layers accelerates development time and democratizes access to state-of-the-art techniques across various domains, including computer vision, natural language processing, and time series analysis.

- **D. Numpy:** NumPy plays a crucial role in deep learning projects, serving as the backbone for numerical computation and data manipulation tasks. Its efficient array operations and broadcasting capabilities enable seamless manipulation of large datasets, facilitating tasks such as data preprocessing, feature extraction, and model evaluation.
- **E.Pandas:** Pandas is an indispensable tool in the realm of deep learning projects, primarily serving as a powerful data manipulation and analysis library in Python. Making it ideal for tasks such as data preprocessing, exploration, and feature engineering in deep learning workflows. Its rich set of functions and methods allow for efficient data cleaning, transformation, and integration, enabling practitioners to prepare diverse datasets for model training with ease.
- **F. CV2:** OpenCV (Open Source Computer Vision Library) plays a pivotal role in deep learning projects, particularly in computer vision applications. As a versatile library, OpenCV provides a wide array of functions and tools for image and video processing. With its robust feature set, OpenCV enables practitioners to perform essential operations like image manipulation, filtering, and feature extraction, laying the groundwork for building and training deep neural networks. Its seamless integration with popular deep learning frameworks like TensorFlow and PyTorch facilitates the development and deployment of end-to-end computer vision.

CHAPTER 5

SYSTEM DESIGN

5.1 SYSTEM IMPLEMENTATION

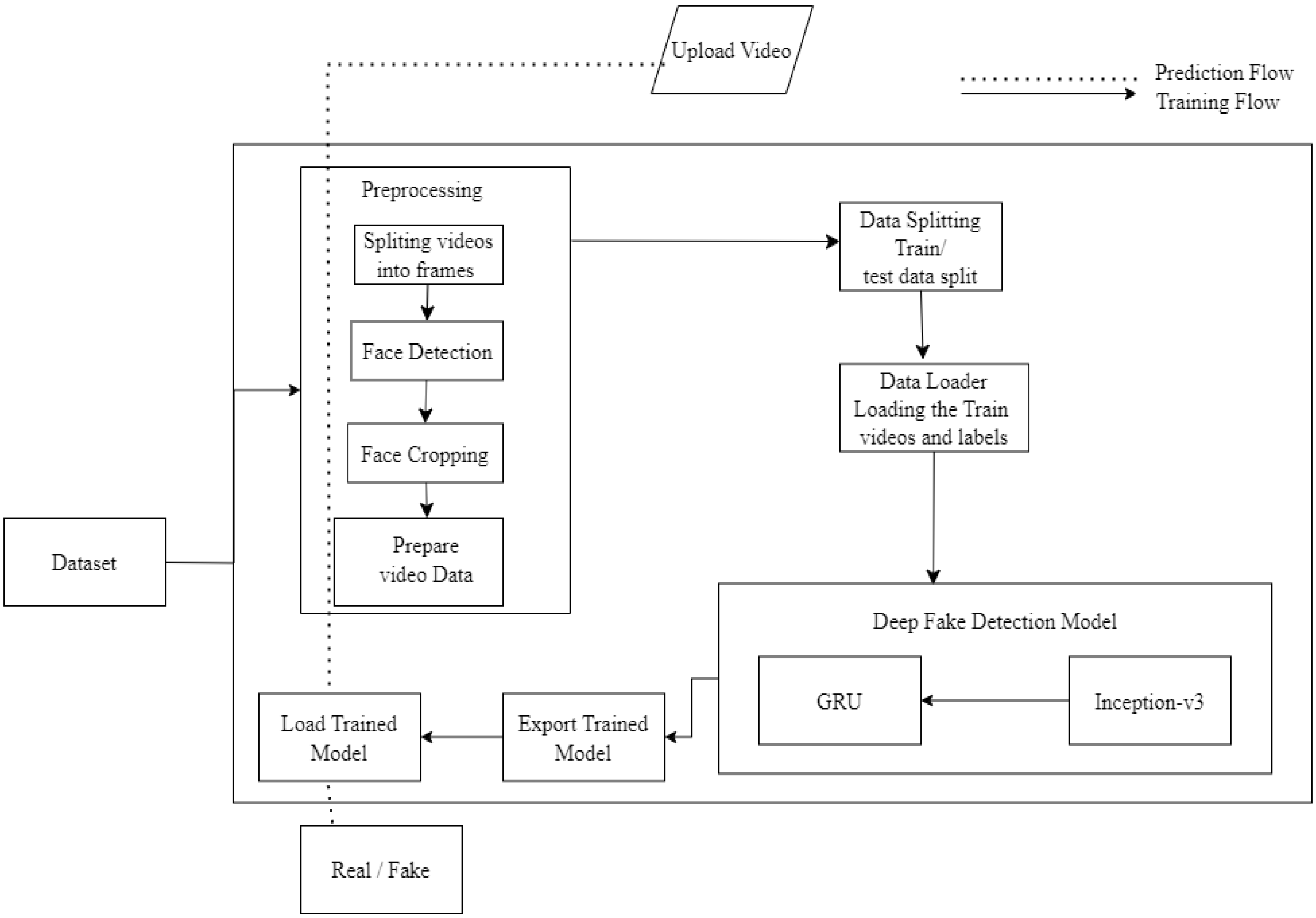


Fig 5.1: System Implementation

5.2. UML DIAGRAMS

UML (Unified Modeling Language) is a standardized graphical modeling language used to visualize, specify, construct, and document the artifacts of a software system. It provides a set of diagrams and notations to represent different aspects of a system's design and architecture.

The UML diagram shown in the image appears to be a type of UML diagram, specifically a use case diagram. Use case diagrams are used to model the functionality and interactions of a system from the perspective of the users or actors.
The "Admin" element represents an actor, which is typically a user, system, or external entity that interacts with the system.

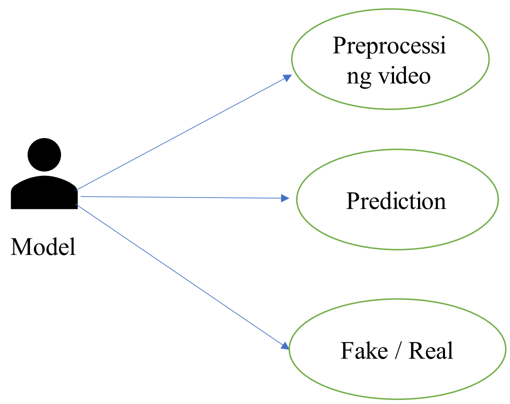
The different elements connected to the "Admin" actor (Analysis, Project, Organization, Tasks, Departments, Chat, Calls, Files) represent use cases, which are the specific functionalities or actions that the system provides to the actor.

The connections between the "Admin" actor and the use cases indicate the relationship between the actor and the system's functionality.

UML diagrams, including use case diagrams, are widely used in software engineering and system design to help understand, analyze, and communicate the requirements, structure, and behavior of a system. They provide a standardized visual language that can be used throughout the software development lifecycle, from requirements gathering to design and implementation.

The connections between the User and the various use cases (Project, Tasks, Chats, Calls, Files) indicate the interactions and relationships between the user and the system's functionalities.

USE CASE DIAGRAM



12

Fig 5.2.2 Admin Use case Diagram

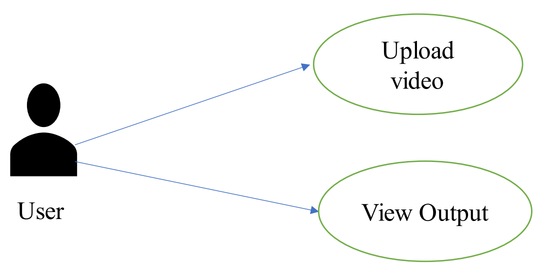


Fig 5.2.1 User Use case Diagram

5.3 DEEPFAKE DETECTION PROCESS

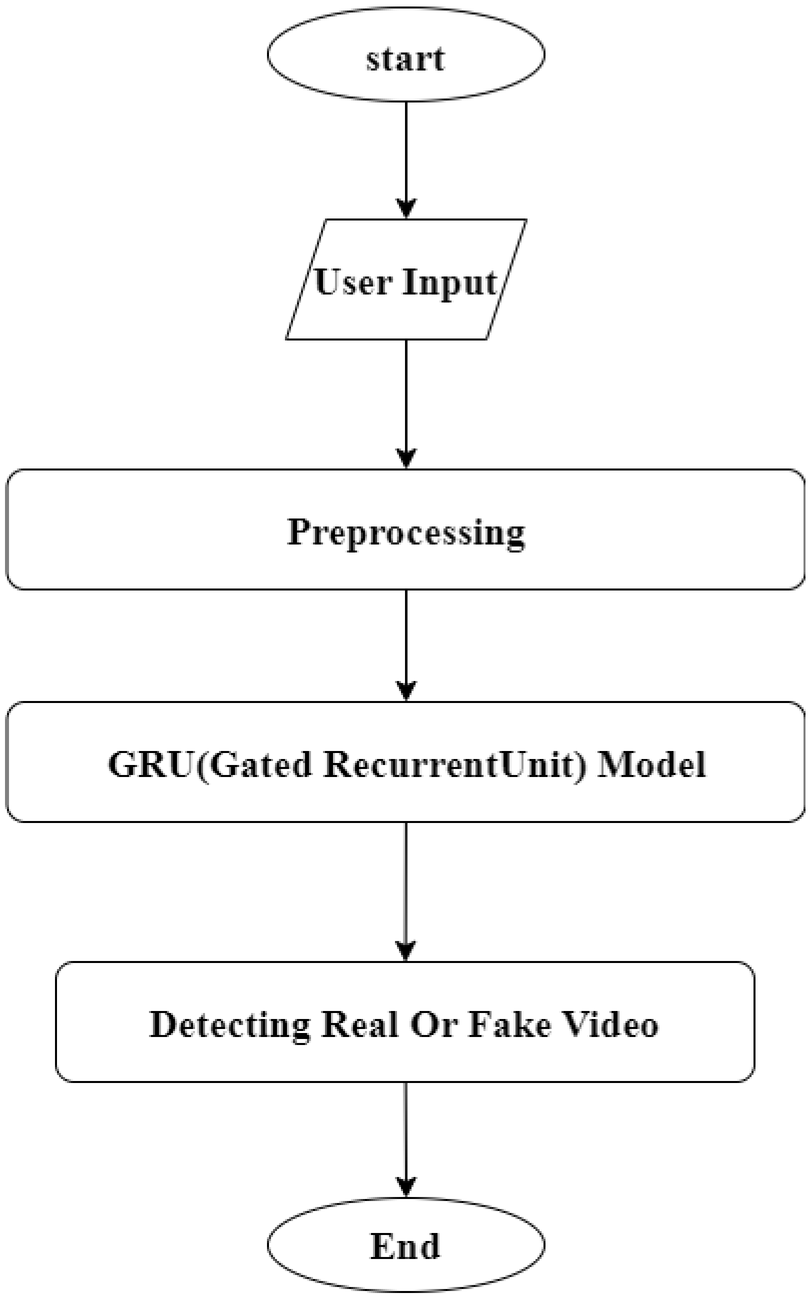


Fig 5.3 Flowchart of deepfake detection model

CHAPTER 6

IMPLEMENTATION

GRU (Gated Recurrent Unit) ALGORITHM:

- The GRU is a type of recurrent neural network (RNN) architecture designed to address the limitations of traditional RNNs, such as the vanishing gradient problem.
- It is particularly effective in modeling sequential data, making it suitable for tasks like time series prediction, natural language processing, and video analysis.
- Update Gate: Controls how much of the previous hidden state to retain and how much of the new state to consider.
- Reset Gate: Determines how much of the previous state should be ignored in computing the new candidate state.
- Candidate Activation: Computes a new candidate state based on the current input and the previous hidden state.
- At each time step, the GRU takes an input vector and the previous hidden state as input.
- The update gate and reset gate are calculated based on the input and previous hidden state.
- Due to their simpler architecture, GRUs are easier to train compared to more complex models.
- The Gated Recurrent Unit (GRU) algorithm offers an efficient and effective solution for modeling sequential data, making it a valuable tool for various machine learning tasks, including deepfake detection.

Code.py

```
pip install -U --upgrade tensorflow
from tensorflow import keras
import matplotlib.pyplot as plt
import tensorflow as tf
import pandas as pd
import numpy as np
import imageio
import cv2
import os
DATA_FOLDER = '/content/deepfake-detection-challenge.zip'
TRAIN_SAMPLE_FOLDER = '/content/train_sample_videos'
TEST_FOLDER = '/content/test_videos'


print(f"Train samples: {len(os.listdir(os.path.join(DATA_FOLDER, TRAIN_SAMPLE_FOLDER)))}")
print(f"Test samples: {len(os.listdir(os.path.join(DATA_FOLDER, TEST_FOLDER)))}")


train_sample_metadata = pd.read_json('/content/train_sample_videos/metadata.json').T
train_sample_metadata.head()


train_sample_metadata.groupby('label')['label'].count().plot(figsize=(15, 5), kind='bar', title='Distribution of Labels in the Training Set')
plt.show()


train_sample_metadata.shape
```

```
fake_train_sample_video = list(train_sample_metadata.loc[train_sample_metadata.label=='FAKE'].sample(3).index)
fake_train_sample_video

def display_image_from_video(video_path):
    capture_image = cv2.VideoCapture(video_path)
    ret, frame = capture_image.read()
    fig = plt.figure(figsize=(10,10))
    ax = fig.add_subplot(111)
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    ax.imshow(frame)

for video_file in fake_train_sample_video:
    display_image_from_video(os.path.join(DATA_FOLDER, TRAIN_SAMPLE_FOLDER, video_file))

real_train_sample_video = list(train_sample_metadata.loc[train_sample_metadata.label=='REAL'].sample(3).index)
real_train_sample_video

for video_file in real_train_sample_video:
    display_image_from_video(os.path.join(DATA_FOLDER, TRAIN_SAMPLE_FOLDER, video_file))

train_sample_metadata['original'].value_counts()[0:5]

def display_image_from_video_list(video_path_list, video_folder=TRAIN_SAMPLE_FOLDER):
```

```
    plt.figure()
    fig, ax = plt.subplots(2,3,figsize=(16,8))
    for i, video_file in enumerate(video_path_list[0:6]):
        video_path = os.path.join(DATA_FOLDER, video_folder,video_file)
        capture_image = cv2.VideoCapture(video_path)
        ret, frame = capture_image.read()
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        ax[i//3, i%3].imshow(frame)
        ax[i//3, i%3].set_title(f"Video: {video_file}")
        ax[i//3, i%3].axis('on')

same_original_fake_train_sample_video = list(train_sample_metadata.loc[train_sample_metadata.original=='atvmxvwyns.mp4'].index)
display_image_from_video_list(same_original_fake_train_sample_video)

test_videos = pd.DataFrame(list(os.listdir(os.path.join(DATA_FOLDER, TEST_FOLDER))), columns=['video'])

test_videos.head()

display_image_from_video(os.path.join(DATA_FOLDER, TEST_FOLDER, test_videos.iloc[2].video))

fake_videos = list(train_sample_metadata.loc[train_sample_metadata.label=='FAKE'].index)

from IPython.display import HTML
from base64 import b64encode
```

```
def play_video(video_file, subset=TRAIN_SAMPLE_FOLDER):

    video_url = open(os.path.join(DATA_FOLDER, subset,video_file),'rb').read()
    data_url = "data:video/mp4;base64," + b64encode(video_url).decode()
    return HTML("""<video width=500 controls><source src="%s" type="video/mp4"></video>""" % data_url)

play_video(fake_videos[10])

IMG_SIZE = 224
BATCH_SIZE = 64
EPOCHS = 150

MAX_SEQ_LENGTH = 20
NUM_FEATURES = 2048

def crop_center_square(frame):
    y, x = frame.shape[0:2]
    min_dim = min(y, x)
    start_x = (x // 2) - (min_dim // 2)
    start_y = (y // 2) - (min_dim // 2)
    return frame[start_y : start_y + min_dim, start_x : start_x + min_dim]


def load_video(path, max_frames=0, resize=(IMG_SIZE, IMG_SIZE)):
    cap = cv2.VideoCapture(path)
    frames = []
```

```
    try:
        while True:
            ret, frame = cap.read()
            if not ret:
                break
            frame = crop_center_square(frame)
            frame = cv2.resize(frame, resize)
            frame = frame[:, :, [2, 1, 0]]
            frames.append(frame)
            if len(frames) == max_frames:
                break
    finally:
        cap.release()
    return np.array(frames)


def build_feature_extractor():
    feature_extractor = keras.applications.InceptionV3(
        weights="imagenet",
        include_top=False,
        pooling="avg",
        input_shape=(IMG_SIZE, IMG_SIZE, 3),
    )
    preprocess_input = keras.applications.inception_v3.preprocess_input

    inputs = keras.Input((IMG_SIZE, IMG_SIZE, 3))
    preprocessed = preprocess_input(inputs)
```

```
    outputs = feature_extractor(preprocessed)
    return keras.Model(inputs, outputs, name="feature_extractor")


feature_extractor = build_feature_extractor()

def prepare_all_videos(df, root_dir):
    num_samples = len(df)
    video_paths = list(df.index)
    labels = df["label"].values
    labels = np.array(labels=='FAKE').astype(np.int64)


    frame_masks = np.zeros(shape=(num_samples, MAX_SEQ_LENGTH), dtype="bool")
    frame_features = np.zeros(
        shape=(num_samples, MAX_SEQ_LENGTH, NUM_FEATURES), dtype="float32"
    )


    for idx, path in enumerate(video_paths):

        frames = load_video(os.path.join(root_dir, path))
        frames = frames[None, ...]


        temp_frame_mask = np.zeros(shape=(1, MAX_SEQ_LENGTH,), dtype="bool")
        temp_frame_features = np.zeros(
```

```
            shape=(1, MAX_SEQ_LENGTH, NUM_FEATURES), dtype="float32"
        )


        for i, batch in enumerate(frames):
            video_length = batch.shape[0]
            length = min(MAX_SEQ_LENGTH, video_length)
            for j in range(length):
                temp_frame_features[i, j, :] = feature_extractor.predict(
                    batch[None, j, :]
                )
            temp_frame_mask[i, :length] = 1  # 1 = not masked, 0 = masked

        frame_features[idx,] = temp_frame_features.squeeze()
        frame_masks[idx,] = temp_frame_mask.squeeze()

    return (frame_features, frame_masks), labels

from sklearn.model_selection import train_test_split

Train_set, Test_set = train_test_split(train_sample_metadata,test_size=0.1,random_state=42,stratify=train_sample_metadata['label'])

print(Train_set.shape, Test_set.shape )


import numpy as np
```

```
train_data, train_labels = prepare_all_videos(Train_set, "train")
test_data, test_labels = prepare_all_videos(Test_set, "test")

print(f"Frame features in train set: {train_data[0].shape}")
print(f"Frame masks in train set: {train_data[1].shape}")

frame_features_input = keras.Input((MAX_SEQ_LENGTH, NUM_FEATURES))
mask_input = keras.Input((MAX_SEQ_LENGTH,), dtype="bool")



def model_r(frame_features_input,mask_input):
    x = keras.layers.GRU(16, return_sequences=True)(
      frame_features_input, mask=mask_input
    )
    x = keras.layers.GRU(8)(x)
    x = keras.layers.Dropout(0.4)(x)
    x = keras.layers.Dense(8, activation="relu")(x)
    output = keras.layers.Dense(1, activation="sigmoid")(x)

    model = keras.Model([frame_features_input, mask_input], output)
    model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])
    return model
model = model_r(frame_features_input,mask_input)
model.summary()
```

```
checkpoint = keras.callbacks.ModelCheckpoint('./.weights.h5', save_weights_only=True, save_best_only=True)

history = model.fit(
    [train_data[0], train_data[1]],
    train_labels,
    validation_data=([test_data[0], test_data[1]],test_labels),
    callbacks=[checkpoint],
    epochs=EPOCHS,
    batch_size=8
  )
from sklearn.metrics import accuracy_score
_, test_accuracy = model.evaluate([test_data[0], test_data[1]], test_labels)
print(f'Test Accuracy: {test_accuracy}')



predictions = model.predict([test_data[0], test_data[1]])



accuracy = accuracy_score(test_labels, predictions.round())
print(f'Accuracy: {accuracy}')


def prepare_single_video(frames):
  frames = frames[None, ...]
  frame_mask = np.zeros(shape=(1, MAX_SEQ_LENGTH,), dtype="bool")
  frame_features = np.zeros(shape=(1, MAX_SEQ_LENGTH, NUM_FEATURES), dtype="float32")


  for i, batch in enumerate(frames):
```

```
        video_length = batch.shape[0]
        length = min(MAX_SEQ_LENGTH, video_length)
        for j in range(length):
            frame_features[i, j, :] = feature_extractor.predict(batch[None, j, :])
        frame_mask[i, :length] = 1  # 1 = not masked, 0 = masked

    return frame_features, frame_mask

def sequence_prediction(path):
    frames = load_video(os.path.join(DATA_FOLDER, TEST_FOLDER,path))
    frame_features, frame_mask = prepare_single_video(frames)
    print(type(model))
    return model.predict([frame_features, frame_mask])[0]



def to_gif(images):
    converted_images = images.astype(np.uint8)
    imageio.mimsave("animation.gif", converted_images, fps=10)
    return embed.embed_file("animation.gif")

test_video = "/content/test_videos/bcbqxhziqz.mp4"
print(f"Test video path: {test_video}")

if(sequence_prediction(test_video)<=0.5):
    print(f'The predicted class of the video is FAKE')
else:
    print(f'The predicted class of the video is REAL')
```

```
play_video(test_video,TEST_FOLDER)



import pickle
model.save("./model.keras")

saved_model  = keras.models.load_model("/content/model.keras")
test_video = "/content/test_videos/aagfhgtpmv.mp4"

def sequence_prediction(path):
    frames = load_video(os.path.join(DATA_FOLDER, TEST_FOLDER,path))
    frame_features, frame_mask = prepare_single_video(frames)

    return saved_model.predict([frame_features, frame_mask])[0]

if(sequence_prediction(test_video)<=0.5):
    print(f'The predicted class of the video is FAKE')
else:
    print(f'The predicted class of the video is REAL')
print(sequence_prediction(test_video))



play_video(test_video,TEST_FOLDER)
```

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
    <title style="color: red";"font-size:200%";>Upload Video</title>

</head>
<body>
    <h1>Upload Video</h1>
    <form action="/predict" method="post" enctype="multipart/form-data">
        <input type="file" name="video">
        <input type="submit" value="Predict">
    </form>
    <div class="video-container">
        <video controls id="uploaded-video">
            <source src="" type="video/mp4">
            Your browser does not support the video tag.
        </video>
    </div>
</div>
<script>
    // JavaScript to display the uploaded video
    const videoInput = document.querySelector('input[name="video"]');
    const video = document.getElementById('uploaded-video');
```

```
    videoInput.addEventListener('change', function(event) {
        const file = event.target.files[0];
        const url = URL.createObjectURL(file);
        video.src = url;
    });
</script>
</body>
</html>
```

Result.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Result</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style1.css') }}">
</head>
<body>
    <h1>Prediction Result</h1>
    <p><center><b>{{ prediction }}</b></center></p>
</body>
</html>
```

Style.css

```
body {
    font-family: Arial, sans-serif;
```

```
    background-color: #f0f0f0;
    margin: 0;
    padding: 0;
    background-image: url('pic.jpeg');
    text-align:center

}

.container {
    max-width: 600px;
    margin: 50px auto;
    background-color: #fff;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

h1 {
    text-align: center;
    margin-bottom: 20px;
}

.upload-btn {
    display: block;
    width: 100%;
    margin-bottom: 10px;
    padding: 10px;
```

```
    border: 1px solid #ccc;
    border-radius: 4px;
    font-size: 16px;
}

.predict-btn {
    display: block;
    width: 100%;
    padding: 10px;
    background-color: #007bff;
    color: #fff;
    border: none;
    border-radius: 4px;
    font-size: 16px;
    cursor: pointer;
    transition: background-color 0.3s;
}

.predict-btn:hover {
    background-color: #0056b3;
}

.prediction {
    text-align: center;
    font-size: 20px;
    margin-top: 20px;
}
```

Style1.css

```
body {
    font-family: Arial, sans-serif;
    background-color: #f0f0f0;
    margin: 0;
    padding: 0;
    background-image: url('pic.jpeg');

}

.container {
    max-width: 600px;
    margin: 50px auto;
    background-color: #fff;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

h1 {
    text-align: center;
    margin-bottom: 20px;
}

.prediction {
```

```
    text-align: center;
    font-size: 24px;
    font-weight: bold;
    color: #32ca0c; /* Blue color for REAL prediction */
}

.fake {
    text-align:center;
    font: size 24px;
    font-weight:bold;
    color: #dc3545; /* Red color for FAKE prediction */
}

.video-container {
    margin-top: 20px;
    text-align: center;
}

video {

    max-width: 100%;
    height: auto;
    border-radius: 8px;
    text-align: center;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
```

Flask.py

```
from flask import Flask, render_template, request
import cv2
import numpy as np
from tensorflow import keras
import os

app = Flask(__name__)

# Define constants
IMG_SIZE = 224
MAX_SEQ_LENGTH = 20
percent_of_expect=0.56
NUM_FEATURES = 2048
MODEL_PATH = "model/model.keras"
#MODEL_PATH="C:\Main Project 1\LSTM.keras"

# Function definitions
def crop_center_square(frame):
    y, x = frame.shape[0:2]
    min_dim = min(y, x)
    start_x = (x // 2) - (min_dim // 2)
    start_y = (y // 2) - (min_dim // 2)
    return frame[start_y : start_y + min_dim, start_x : start_x + min_dim]

def load_video_n(path, max_frames=0, resize=(IMG_SIZE, IMG_SIZE)):
    cap = cv2.VideoCapture(path)
```

```
    frames = []
    try:
        while True:
            ret, frame = cap.read()
            if not ret:
                break
            frame = crop_center_square(frame)
            frame = cv2.resize(frame, resize)
            frame = frame[:, :, [2, 1, 0]]
            frames.append(frame)
            if len(frames) == max_frames:
                break
    finally:
        cap.release()
    return np.array(frames)


def prepare_single_video_n(frames):
    frames = frames[None, ...]
    frame_mask = np.zeros(shape=(1, MAX_SEQ_LENGTH,), dtype="bool")
    frame_features = np.zeros(shape=(1, MAX_SEQ_LENGTH, NUM_FEATURES), dtype="float32")

    for i, batch in enumerate(frames):
        video_length = batch.shape[0]
        length = min(MAX_SEQ_LENGTH, video_length)
        for j in range(length):
            frame_features[i, j, :] = feature_extractor.predict(batch[None, j, :])
```

```
        frame_mask[i, :length] = 1  # 1 = not masked, 0 = masked

    return frame_features, frame_mask

def build_feature_extractor():
    feature_extractor = keras.applications.InceptionV3(
        weights="imagenet",
        include_top=False,
        pooling="avg",
        input_shape=(IMG_SIZE, IMG_SIZE, 3),
    )
    preprocess_input = keras.applications.inception_v3.preprocess_input

    inputs = keras.Input((IMG_SIZE, IMG_SIZE, 3))
    preprocessed = preprocess_input(inputs)

    outputs = feature_extractor(preprocessed)
    return keras.Model(inputs, outputs, name="feature_extractor")

def sequence_prediction(path):
    frames = load_video_n(path)
    frame_features, frame_mask = prepare_single_video_n(frames)
    percent= saved_model.predict([frame_features, frame_mask])[0]
    print(percent)
    return percent

# Load the saved model
```

```
MODEL_PATH="C:\Main Project 1\model.keras"
saved_model = keras.models.load_model(MODEL_PATH)
feature_extractor = build_feature_extractor()

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    if request.method == 'POST':
        # Get the path of the uploaded video
        video_file = request.files['video']
        video_path = "uploads/" + video_file.filename
        video_file.save(video_path)

        # Perform sequence prediction
        result = sequence_prediction(video_path)



        # Delete the uploaded video after prediction
        os.remove(video_path)

        # Determine the prediction class
        prediction_class = "FAKE" if result <= percent_of_expect else "REAL"
        print(prediction_class)
```

```
        # Render the result template with the prediction
        return render_template('result.html', prediction=prediction_class)
        #print(prediction_class)

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

CHAPTER 7
OUTPUT SCREENS

Fig 7.1 webpage preview

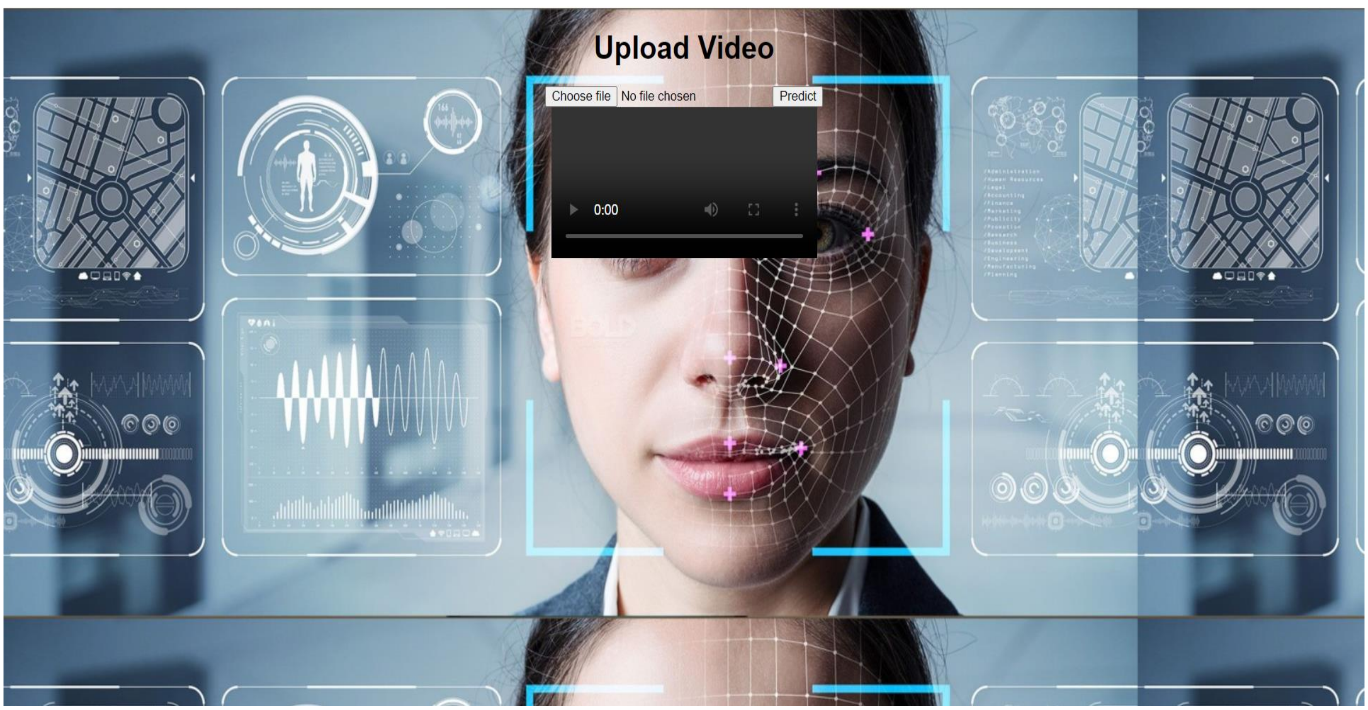


Fig 7.1.1 Video uploaded page

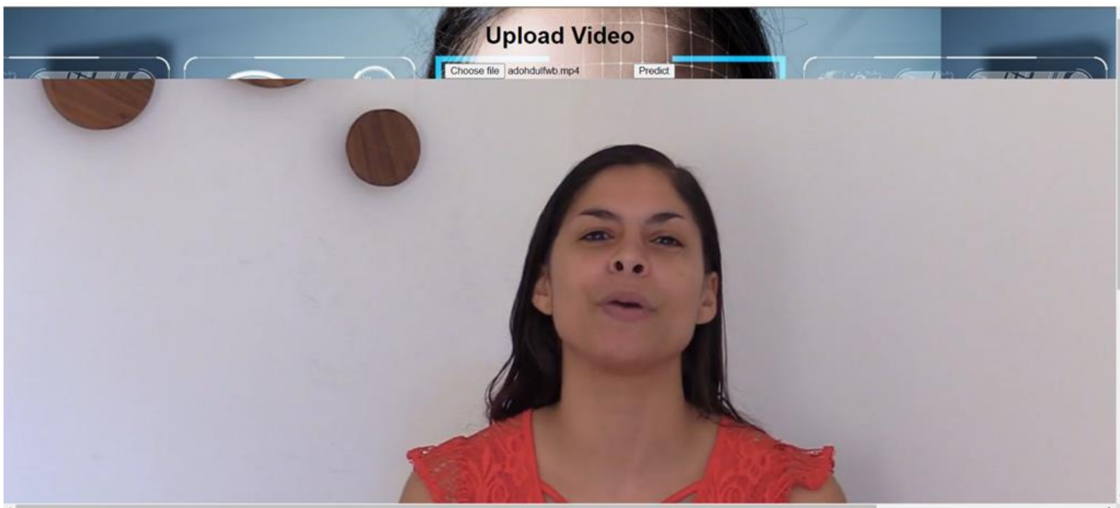


Fig 7.1.2 Result page

CHAPTER 8

SYSTEM TESTING

SYSTEM TESTING

Unit Testing:

While it's not common for coding and unit testing to be distinct stages, unit testing usually occurs within a combined phase of code and unit testing in the software lifecycle. Evaluate your strategy and schedule. We'll conduct manual field testing and develop thorough functional tests.

System Testing:

System testing analyzes the system's structure and behavior to verify its overall functionality. Conducted independently of the development team, system testing assesses the system's effectiveness. It encompasses comprehensive testing based on functional requirement specifications, system requirement specifications, or both.

Integrating Testing:

Integration testing of software minimizes errors caused by interface issues by progressively combining and testing two or more integrated software components on a unified platform. The objective of integration testing is to ensure that components or software applications, whether within a software system or utilized across an entire organization, seamlessly collaborate.

8.1 TEST CASES

Test Case Id	Test Case Condition	Expected Output	Actual Output	Pass/Fail
T01	Video Upload	Should upload video from the folder and detect video is fake	Uploading the video and detecting that the video is fake	PASSED
T02	Video Upload	Should upload video from the folder and detect video is real	Uploading the video and detecting that the video is real	PASSED
T03	Press predict button without selecting video	Gives permission error from uploads section	Gives permission error from uploads section	PASSED

CHAPTER 9

CONCLUSION AND FUTURE SCOPE

CONCLUSION AND FUTURE SCOPE

9.1 CONCLUSION:

This project involves developing a deep learning solution to detect deepfake videos. Using neural networks such as Gated Recurrent Units (GRUs) and pre-trained convolutional models like InceptionV3, we built a strong system capable of identifying manipulated videos. The GRU model achieved an accuracy of 80% in distinguishing between real and manipulated videos.

9.2 FUTURE SCOPE:

- There's always room for improvement in any system, especially when it's built using cutting-edge technology and has promising future prospects
- At present, the algorithm only detects facial deepfakes, but there' potential to enhance it for detecting full-body deepfakes.

REFERENCES

- Tewari, A., Zollhoefer, M., Bernard, F., Garrido, P., Kim, H., Perez, P., and Theobalt, C. (2018).High-delity monocular face reconstruction based on an unsupervised model-based face autoencoder.IEEE Transactions on Pattern Analysis and Machine Intelligence.
- Guo, Y., Jiao, L., Wang, S., Wang, S., and Liu, F. (2017). Fuzzy sparse autoencoder framework for single image per person face recognition. IEEE Transactions on Cybernetics, 48(8),2402-2415.
- Liu, F., Jiao, L., and Tang, X. (2019). Task-oriented GAN for PolSAR image classication and clustering. IEEE transactions on Neural Networks and Learning Systems, 30(9),2707-2719.
- Lyu, S. (2018, August 29). Detecting deepfake videos in the blink of an eye.Retrievedfrom
- http://theconversation.com/detecting-deepfake-videos-in-the-blink-of-an-eye-101072
- Chesney, R., and Citron, D. (2019). Deepfakes and the new disinformation war: The coming ageof post-truth geopolitics. Foreign A airs, 98,147
- 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
- Rossler, A., Cozzolino, D., Verdoliva, L., Riess, C., Thies, J. and Nießner, M. (2019) Faceforensics++: Learning to Detect Manipulated Facial Images. Proceedings of the IEEE/CVF International Conference on Computer Vision, Seoul, 27-28 October 2019, 1-11. https://doi.org/10.1109/ICCV.2019.00009
- Westerlund, M. (2019) The Emergence of Deepfake Technology: A Review. Technology Innovation Management Review, 9, 40-53. https://doi.org/10.22215/timreview/1282
- Agarwal, Shruti et al. “Watch Those Words: Video Falsification Detection Using Word-Conditioned Facial Motion.” 2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) (2021): 4699-4708.