CHAPTER 1

INTRODUCTION

In the world of ever-growing social media platforms, Deepfakes have emerged as a major threat of AI. Deep Fakes are realistic face-swapped videos that can be used to create political distress, fake terrorism, revenge porn, blackmail. Deepfake is a term for videos and presentations enhanced by artificial intelligence and other modern technology to present falsified results. One of the best examples of deepfakes involves the use of image processing to produce video of celebrities, politicians or others saying or doing things that they never actually said or did.

CHAPTER 2

PROBLEM DEFINITION

The creation of realistic manipulations of digital images and videos has been possible for decades, thanks to visual effects. However, with recent technological advancements in deep learning, the creation of AI-synthesized media, commonly known as deep fakes, has become easier and more realistic.

Unfortunately, detecting deep fakes has proven to be a challenging task. The misuse of deep fakes has resulted in instances of political tension, fake terrorism events, revenge porn, and blackmail. Therefore, it is essential to develop effective methods for detecting deep fakes and preventing their dissemination through social media platforms

CHAPTER 3

LITERATURE SURVEY

3.1. LITERATURE REVIEW

S.No	PAPER TITLE & PUBLICATION DETAILS	NAME OF THE AUTHORS	TECHNICAL IDEAS / ALGORITHMS USED IN THE PAPER & ADVANTAGES	SHORTFALLS/DISADVANTAGES & SOLUTION PROVIDED BY THE PROPOSED SYSTEM
1	MesoNet: a Compact Facial Video Forgery Detection Network, 2018	Darius Afchar, Vincent Nozick, Junichi Yamagishi, Isao Echizen	Convolutional Neural Networks	Only datasets provided as open source, no real time implementation
2	In Ictu Oculi: Exposing AI Generated Fake Face Videos by Detecting Eye Blinking	Yuezun Li, Ming-Ching Chang and Siwei Lyu	Usage of eye blinking to detect the fakeness	Teeth alignment, wrinkles on the face, and incorrect placement of eyebrows are not considered.

3	Exposing DeepFake Videos By Detecting Face Warping Artifacts, 2019.	Yuezun Li, Siwei Lyu	Comparison based on surroundings.	Analysis of video frames is not done.

Tab.3.1 - Literature Survey

3.2 PROPOSED SOLUTION

The proposed solution is a pipeline for efficient detection of forged video using deepfake detection technology for face identity swap. After preprocessing, the original video-level dataset is first simplified to image-level and then faces are cropped using the dlib library. The real face is labeled as 0, and the fake face is labeled as 1, so that the detection results of the image to be detected are displayed with a score of 0 to 1. The closer the score is to 0, the closer the image is to the real; the closer the score is to 1, the closer the image is to fake. This greatly reduces the workload of the detection process and improves detection efficiency.

The video-level dataset used is the DeepFake Detection challenge dataset from Kaggle (4.7 GB) which contains 800 videos (both real and fake), which is widely used in the detection of various fake videos, the training and evaluation of detection models are performed on these large-scale fake face video datasets.

CHAPTER 4

REQUIREMENTS SPECIFICATION

Requirements specification is a specification of software requirements and hardware requirements required to do the project.

4.1 Hardware Requirements Specification

Hardware Requirements are the hardware resources that are needed to do the project work. These resources are a computer resource that provides functions and services to do the project. Hardware resources required for our project are shown below.

- Processor : Intel Core i5/Mac M1 or above
- RAM : >=8GB
- Hard disk : Minimum 10 GB
- Google Drive : Minimum 5 GB free space

4.2 Software Requirements Specification

Software Requirements are the software resources that are needed to do the project work. These resources are installed on a computer in order to provide functions, services, hardware accessing capabilities to do the project.

In our project we used the following software resources.

- Operating System: Windows, MacOS or Linux operating system
- Code Environment: Google Colab.
- Storage: Google Drive.
- Browser: Chrome, Safari or Edge.
- Jupyter Environment: Provided by Colab.
- Github: For version control.

4.3 Functional Requirements:

Functional requirements specify a function that system or a system component must be able to perform.

- Video Input: The system should be able to accept video files as input for deepfake detection.
- Image Extraction: The system should extract frames from the input videos and detect faces in each frame using face detection algorithms.
- Data Collection: The system should collect and store the extracted images along with their corresponding labels (real or fake) for training and evaluation purposes.
- Training: The system should train a deep learning model on the collected data to learn the features and patterns indicative of real and fake images.
- Model Evaluation: The system should evaluate the trained model's performance using appropriate evaluation metrics, such as accuracy, precision, recall, and F1 score.
- Deepfake Classification: The system should classify each detected face as real or fake based on the trained model's predictions.
- Probability Assessment: The system should provide a probability score or confidence level for each classification, indicating the likelihood of the detected face being a deepfake.
- Visualization: The system should visualize the detection results by drawing bounding boxes around the detected faces and displaying the probability scores.
- Threshold Configuration: The system should allow the configuration of a threshold value to determine the classification of a video as real, fake, or possibly fake based on the cumulative probabilities of the detected faces.
- Model Management: The system should provide functionalities to save, load, update, and manage the trained deep learning model for future use.

- System Performance: The system should handle video processing and deepfake detection efficiently to ensure real-time or timely detection without significant delays or resource constraints.

4.4 Non-Functional Requirements:

- Reliability: Database updating should follow transaction processing to avoid data inconsistency.
- Availability: The project will be deployed on a public shared server so it will be available all the time and will be accessible anywhere in the world using the internet.
- Security: We have implemented a lot of security mechanisms to avoid having the system hacked by the outer world.
- Maintainability: It is very easy to maintain the system. The system has been developed on google colab so anyone who has the knowledge of colab, can easily maintain the system.
- Portability: Yes this system is portable and we can switch the servers very easily.
- Browser Compatibility: as google colab is a web based development environment, browsers such as Safari, Google Chrome, Edge and Mozilla should be compatible.

CHAPTER 5

SYSTEM DESIGN

5.1 Data Flow Diagram

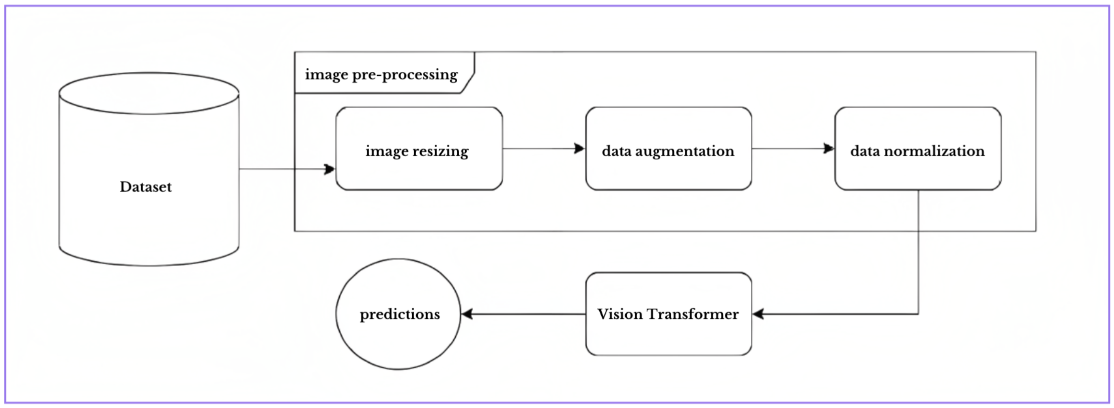


Fig.5.1 - Data Flow diagram

5.2 Use case Diagram

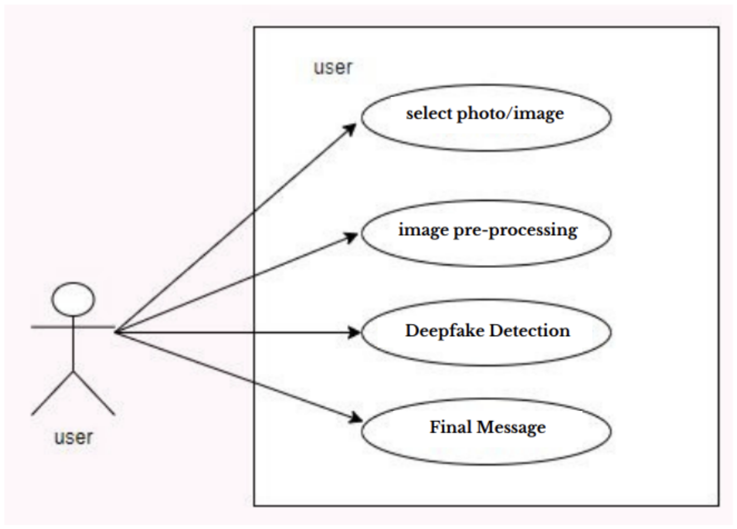


Fig.5.2 - Use Case diagram

5.3 Module Description

1. **Input data module**: involves importing data containing real and fake videos.
2. **Image processing module:** involves using OpenCV and dlib to detect and extract faces from video frames, resizing them to a consistent size, converting them to NumPy arrays, and normalizing the pixel values for further processing in the deep learning model.
3. **Data Splitting and Normalization:** In this module, the processed data is split into training and validation sets using train_test_split(), the labels are converted to categorical format using one-hot encoding, and the input data is reshaped to the desired shape for the deep learning model.
4. **Video Classification and Result Analysis:** This module involves randomly selecting frames from a video, analyzing the predicted probabilities to determine the classification and fakeness metrics, displaying the video frames with bounding boxes and probability labels, and calculating the average, maximum, and minimum probabilities of fakeness for the video.

CHAPTER 6

IMPLEMENTATION

6.1 Tools and Technologies Used

Google Colab: Google Colab is a cloud-based development platform provided by Google that allows users to write and execute Python code through their web browser. It provides a Jupyter notebook environment where users can create and share interactive code notebooks. Colab is particularly useful in the field of Digital Image Processing (DIP) as it offers several advantages.

Google Drive: The code mounts Google Drive to access and save files and folders.

dlib: A library that provides tools and algorithms for computer vision tasks, including face detection and facial landmark estimation.

opencv: OpenCV, or Open Source Computer Vision Library, is an open-source computer vision and machine learning software library. It provides a comprehensive set of functions and algorithms for image and video processing, object detection and recognition, feature extraction, and more.

TensorFlow: A popular deep learning framework used for building and training neural networks. It provides high-level APIs for defining and training models.

numpy: A library for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions.

PIL (Pillow): A Python imaging library that provides functions for opening, manipulating, and saving various image file formats.

keras: A deep learning library that provides a user-friendly interface for building and training neural networks. It is integrated with TensorFlow and used for model creation and training.

matplotlib: A plotting library for creating visualizations, such as line plots, scatter plots, and histograms. It is used to plot the accuracy and loss curves during model training.

seaborn: A data visualization library based on matplotlib. It provides a high-level interface for creating informative and visually appealing statistical graphics.

pandas: A data manipulation and analysis library. It provides data structures and functions for efficiently handling and analyzing structured data, such as data frames.

6.2 Algorithms / Methodologies Used

Face Detection using dlib: The code utilizes the dlib library's frontal face detector, which implements the Histogram of Oriented Gradients (HOG) algorithm combined with a linear classifier (Support Vector Machine) to detect faces in images and videos. This algorithm is effective in locating the bounding boxes around faces.

InceptionResNetV2: The code uses the InceptionResNetV2 architecture as the backbone of the deep learning model. InceptionResNetV2 is a state-of-the-art convolutional neural network (CNN) architecture that combines the ideas of the Inception network and residual connections. It has proven to be successful in various computer vision tasks, including image classification and feature extraction.

Transfer Learning: The code applies transfer learning by utilizing the pre-trained weights of the InceptionResNetV2 model, which were trained on a large dataset (ImageNet). By leveraging the pre-trained model, the code can benefit from the learned features and patterns in the initial layers of the network, allowing for better performance even with limited training data.

Data Augmentation: Although not explicitly shown in the provided code snippet, data augmentation is a common technique used in deep learning for increasing the diversity of the training data. It involves applying random transformations (e.g., rotation, zoom, flip) to the input images, resulting in an augmented dataset. Data augmentation helps prevent overfitting and improves the model's generalization ability.

Binary Cross-Entropy Loss: The code uses the binary cross-entropy loss function as the objective during model training. This loss function is suitable for binary classification problems, where the model is trained to classify each image as either real or fake.

Adam Optimizer: The code employs the Adam optimizer for training the deep learning model. Adam is an adaptive learning rate optimization algorithm that combines the benefits of AdaGrad and RMSProp. It adjusts the learning rate dynamically based on the gradients of the model parameters, allowing for efficient convergence during training.

Evaluation Metrics: The code utilizes accuracy and loss metrics to evaluate the performance of the trained model. Accuracy represents the percentage of correctly classified samples, while loss measures the discrepancy between the predicted and true labels. These metrics help assess the model's performance and monitor its convergence during training.

6.3 Implementation Steps

Data Collection: Videos containing both real and fake samples are collected. The metadata of the videos, including the labels (REAL or FAKE), are stored in a JSON file.

Image Extraction: Each video is processed frame by frame. The dlib library is used to detect faces in each frame. The detected face regions are cropped and saved as individual images. For real videos, the images are saved in the "real" folder, and for fake videos, the images are saved in the "fake" folder.

Data Preprocessing: The saved images are loaded and preprocessed. The pixel values are normalized, and the images are reshaped to the desired input shape for the model.

Model Training: The InceptionResNetV2 model is created and trained on the preprocessed images. The model is compiled with the binary cross-entropy loss function and the Adam optimizer. The training data is split into training and validation sets.

Model Evaluation: The model's performance is evaluated using accuracy and loss metrics. The training and validation accuracy and loss curves are plotted. Additionally, a confusion matrix is generated to analyze the model's performance in classifying real and fake samples.

Model Saving: The trained model is saved for future use.

Deep Face Detection on a Video: A new video is loaded, and deepfake detection is performed on each frame. The dlib library is used to detect faces in each frame, and the detected face regions are passed through the trained model. The model predicts the probability of fakeness for each face, and the results are displayed by drawing bounding boxes around the detected faces and showing the probability of fakeness.

Video Classification: The video is classified based on the detection results. If any face in the video has a probability of fakeness equal to 1.0 (100%), the video is classified as FAKE. If the number of frames with a probability exceeding 0.5 (50%) falls within a certain range (3-5 frames), the video is classified as POSSIBLY FAKE. If more than 5 frames have a probability exceeding 0.5, the video is classified as FAKE. Otherwise, the video is classified as REAL.

CHAPTER 7

SYSTEM TESTING

Test Case	Video Type	Expected Result	Result
1	No Video	Error Saying no input found	Successful
2	Real video with no deep fake content	Not a Deep Fake	Successful
3	Real video with very less deep fake content	Possibly a DeepFake	Successful
4	Fake video with no real content	Fake	Successful
5	Fake video with some real content	Possibly a Fake	Successful
6	Fake video with high-quality deep fake content	Fake	Successful (in some instances)
7	Fake video with low-quality deep fake content	Fake	Successful

Tab.7.1 - Test Cases

CHAPTER 8

CONCLUSION AND FUTURE SCOPE

8.1 Conclusion:

The implemented **DeepDetect - Deepfake Detection** system utilizes the **InceptionResNetV2** architecture to analyze videos, extract images, train a deep learning model, and accurately classify videos as real, fake, or possibly fake based on the probability of fakeness, demonstrating its potential as a reliable solution for detecting deep fake content.

8.2 Future Scope:

User-Friendly Interaction: The web interface would enable users to easily upload videos or provide video URLs for analysis without the need for complex command-line operations. Users can have a seamless and intuitive experience by interacting with the system through a web-based graphical user interface (GUI).

Dataset Expansion: Increase the size and diversity of the training dataset by collecting more real and fake videos with various deepfake techniques, lighting conditions, camera angles, and subjects. This can improve the model's generalization and ability to detect a wider range of deep fakes.

Explainability and Interpretability: Develop techniques to provide explanations or visualizations for the model's predictions. This can help users understand the factors influencing the classification decisions, increase trust in the system, and facilitate human verification or auditing.

Real-Time Processing: Optimize the system to perform real-time deepfake detection by leveraging hardware acceleration (e.g., GPUs or specialized AI chips) and implementing efficient algorithms for face detection and classification.

APPENDICES

A. SAMPLE CODE

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

import os
os.chdir('/content/drive/MyDrive/deepfake-detection-master')

import dlib
import cv2
import os
import re
import json
from pylab import *
from PIL import Image, ImageChops, ImageEnhance

train_frame_folder = 'train_sample_videos'
with open(os.path.join(train_frame_folder, 'metadata.json'), 'r') as file:
    data = json.load(file)
list_of_train_data = [f for f in os.listdir(train_frame_folder) if f.endswith('.mp4')]
detector = dlib.get_frontal_face_detector()
for vid in list_of_train_data:
    count = 0
    cap = cv2.VideoCapture(os.path.join(train_frame_folder, vid))
    frameRate = cap.get(5)
    while cap.isOpened():
        frameId = cap.get(1)
        ret, frame = cap.read()
        if ret != True:
            break
        if frameId % ((int(frameRate)+1)*1) == 0:
            face_rects, scores, idx = detector.run(frame, 0)
            for i, d in enumerate(face_rects):
                x1 = d.left()
                y1 = d.top()
                x2 = d.right()
                y2 = d.bottom()
                crop_img = frame[y1:y2, x1:x2]
                if data[vid]['label'] == 'REAL':

cv2.imwrite('/content/drive/MyDrive/deepfake-detection-master/dataset/real/'+vid.split('.'
)[0]+'_'+str(count)+'.png', cv2.resize(crop_img, (128, 128)))
                elif data[vid]['label'] == 'FAKE':

cv2.imwrite('/content/drive/MyDrive/deepfake-detection-master/dataset/fake/'+vid.split('.'
)[0]+'_'+str(count)+'.png', cv2.resize(crop_img, (128, 128)))
                count+=1

import os
import cv2
import json
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sn
import pandas as pd
from tensorflow.keras.preprocessing.image import ImageDataGenerator, img_to_array,
load_img
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

input_shape = (128, 128, 3)
data_dir = '/content/drive/MyDrive/deepfake-detection-master/dataset'

real_data = [f for f in os.listdir(data_dir+'/real') if f.endswith('.png')]
fake_data = [f for f in os.listdir(data_dir+'/fake') if f.endswith('.png')]

X = []
Y = []

for img in real_data:
    X.append(img_to_array(load_img(data_dir+'/real/'+img)).flatten() / 255.0)
    Y.append(1)
for img in fake_data:
    X.append(img_to_array(load_img(data_dir+'/fake/'+img)).flatten() / 255.0)
    Y.append(0)

Y_val_org = Y

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

#Reshape
X = X.reshape(-1, 128, 128, 3)

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

from tensorflow.keras.applications import InceptionResNetV2
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import InputLayer
from tensorflow.keras.layers import GlobalAveragePooling2D
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import Model
from tensorflow.keras import optimizers
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping
from tensorflow.keras.optimizers import legacy

googleNet_model = InceptionResNetV2(include_top=False, weights='imagenet',
input_shape=input_shape)
googleNet_model.trainable = True
model = Sequential()
model.add(googleNet_model)
model.add(GlobalAveragePooling2D())
model.add(Dense(units=2, activation='softmax'))
model.compile(loss='binary_crossentropy',
        optimizer=legacy.Adam(lr=1e-5, beta_1=0.9, beta_2=0.999, epsilon=None,
decay=0.0, amsgrad=False),
        metrics=['accuracy'])
model.summary()

#Currently not used
early_stopping = EarlyStopping(monitor='val_loss',
                     min_delta=0,
                     patience=2,
                     verbose=0, mode='auto')
```

```
EPOCHS = 20
BATCH_SIZE = 100
history = model.fit(X_train, Y_train, batch_size = BATCH_SIZE, epochs = EPOCHS,
validation_data = (X_val, Y_val), verbose = 1)

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 4))
t = f.suptitle('Pre-trained InceptionResNetV2 Transfer Learn with Fine-Tuning & Image
Augmentation Performance ', fontsize=12)
f.subplots_adjust(top=0.85, wspace=0.3)

epoch_list = list(range(1,EPOCHS+1))
ax1.plot(epoch_list, history.history['accuracy'], label='Train Accuracy')
ax1.plot(epoch_list, history.history['val_accuracy'], label='Validation Accuracy')
ax1.set_xticks(np.arange(0, EPOCHS+1, 1))
ax1.set_ylabel('Accuracy Value')
ax1.set_xlabel('Epoch #')
ax1.set_title('Accuracy')
l1 = ax1.legend(loc="best")

ax2.plot(epoch_list, history.history['loss'], label='Train Loss')
ax2.plot(epoch_list, history.history['val_loss'], label='Validation Loss')
ax2.set_xticks(np.arange(0, EPOCHS+1, 1))
ax2.set_ylabel('Loss Value')
ax2.set_xlabel('Epoch #')
ax2.set_title('Loss')
l2 = ax2.legend(loc="best")

#Output confusion matrix
def print_confusion_matrix(y_true, y_pred):
   y_pred_classes = np.argmax(y_pred, axis=1)  # Convert predicted probabilities to class
labels
```

```
    cm = confusion_matrix(y_true, y_pred_classes)
    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) # for label size
    sn.heatmap(df_cm, annot=True, annot_kws={"size": 16}) # font size
    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()

print_confusion_matrix(Y_val_org, model.predict(X))  # Use predict method instead of
predict_classes

model.save('/content/drive/MyDrive/deepfake-detection-master/model/deepfake-detectio
n-model.h5')

import tensorflow as tf
import dlib
import cv2
import os
import numpy as np
from PIL import Image, ImageChops, ImageEnhance
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import img_to_array, load_img
```

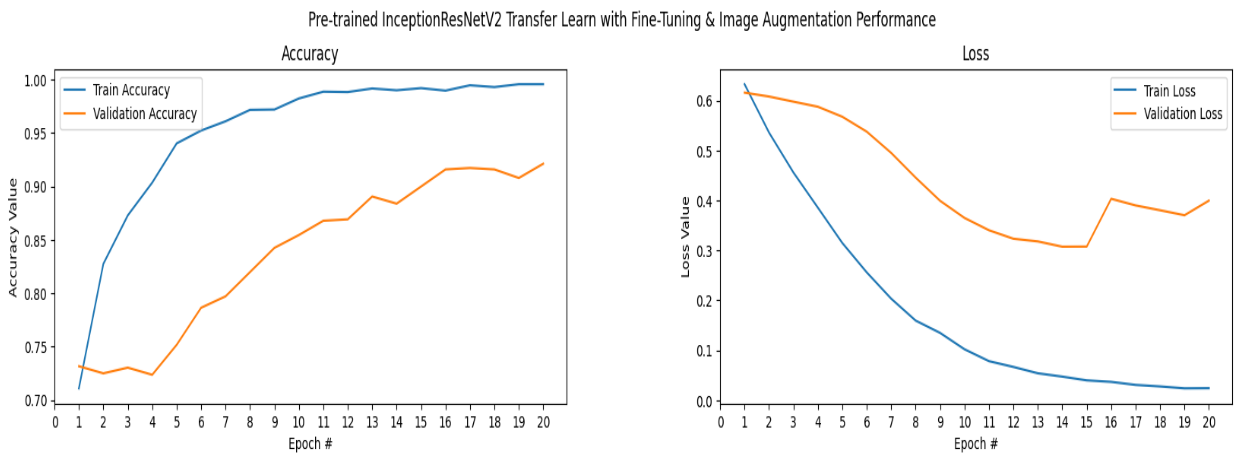
```
tf.__version__

model =
load_model('/content/drive/MyDrive/deepfake-detection-master/model/deepfake-detectio
n-model.h5')

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

import numpy as np
import random
import matplotlib.pyplot as plt

input_shape = (128, 128, 3)
pr_data = []
detector = dlib.get_frontal_face_detector()
cap = cv2.VideoCapture('/content/sample_data/CCqaaV-vtmtmDifI.mp4')
frameCount = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
frameNumbers = random.sample(range(frameCount), 5) # Select 5 random frame
numbers
scores = []

for frameId in frameNumbers:
    cap.set(cv2.CAP_PROP_POS_FRAMES, frameId)
    ret, frame = cap.read()
    if ret != True:
        break
    face_rects, _, _ = detector.run(frame, 0)
    for d in face_rects:
        x1, y1, x2, y2 = d.left(), d.top(), d.right(), d.bottom()
        crop_img = frame[y1:y2, x1:x2]
        data = img_to_array(cv2.resize(crop_img, (128, 128))).flatten() / 255.0
        data = data.reshape(-1, 128, 128, 3)
        score = model.predict(data)[0][1] # probability of being fake
        scores.append(score)
        # Draw a rectangle around the detected face
        cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
        # Add a label with the probability of fakeness to the top of the face frame
        cv2.putText(frame, f'Fakeness: {score:.2f}', (x1, y1-10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
    # Display the video frame with the detected faces and probability of fakeness
    plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
    plt.show()

# Calculate the average probability of the video being fake
average_score = np.mean(scores)

# Calculate the percentage of fakeness
percentage_fakeness = average_score * 100

# Find the maximum and minimum probability of fakeness
max_score = np.max(scores)
min_score = np.min(scores)

print("The percentage of fakeness is:", percentage_fakeness)
print("The maximum probability of fakeness is:", max_score)
print("The minimum probability of fakeness is:", min_score)

import numpy as np
import random
import matplotlib.pyplot as plt

input_shape = (128, 128, 3)
pr_data = []
detector = dlib.get_frontal_face_detector()
cap = cv2.VideoCapture('/content/sample_data/elon.mp4')
frameCount = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
frameNumbers = random.sample(range(frameCount), 15) # Select 5 random frame
numbers
scores = []

# Set the fake bias
fake_bias = 0.1
```

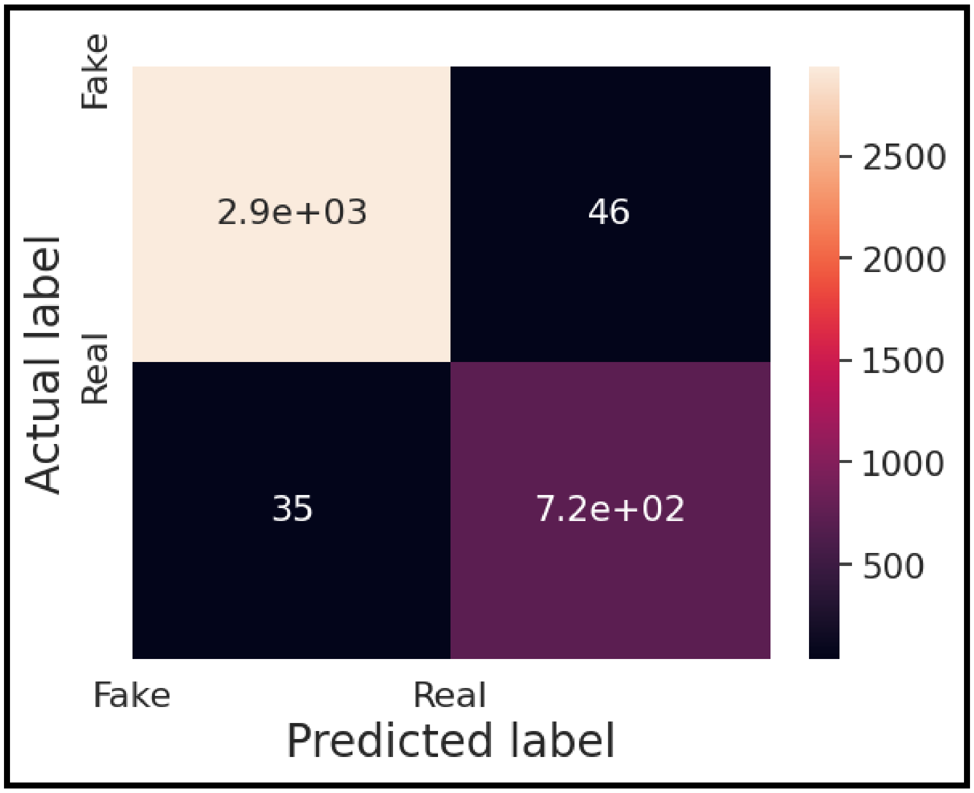
```
for frameId in frameNumbers:
    cap.set(cv2.CAP_PROP_POS_FRAMES, frameId)
    ret, frame = cap.read()
    if ret != True:
        break
    face_rects, _, _ = detector.run(frame, 0)
    for d in face_rects:
        x1, y1, x2, y2 = d.left(), d.top(), d.right(), d.bottom()
        crop_img = frame[y1:y2, x1:x2]
        data = img_to_array(cv2.resize(crop_img, (128, 128))).flatten() / 255.0
        data = data.reshape(-1, 128, 128, 3)
        score = model.predict(data)[0][1] # probability of being fake
        # Apply the bias to the fake score
        score = score + fake_bias if score < 0.5 else score - fake_bias
        scores.append(score)
        # Draw a rectangle around the detected face
        cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
        # Add a label with the probability of fakeness to the top of the face frame
        cv2.putText(frame, f'Fakeness: {score:.2f}', (x1, y1-10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
    # Display the video frame with the detected faces and probability of fakeness
    plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
    plt.show()

# Calculate the biased average probability of the video being fake
average_score = np.mean(scores)

# Calculate the percentage of fakeness
percentage_fakeness = average_score * 100
```

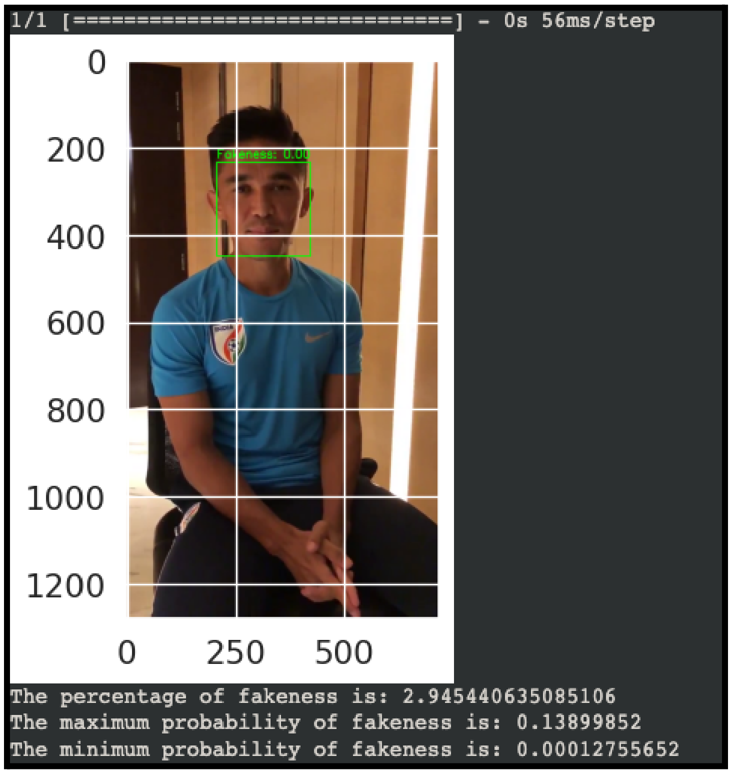
```
# Find the maximum and minimum probability of fakeness
max_score = np.max(scores)
min_score = np.min(scores)

print("The percentage of fakeness is:", percentage_fakeness)
print("The maximum probability of fakeness is:", max_score)
print("The minimum probability of fakeness is:", min_score)

import numpy as np
import random
import matplotlib.pyplot as plt

input_shape = (128, 128, 3)
pr_data = []
detector = dlib.get_frontal_face_detector()
cap = cv2.VideoCapture('/content/mf.mp4')
frameCount = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
frameNumbers = random.sample(range(frameCount), 20) # Select 20 random frame
numbers
fake_frame_count = 0  # Initialize fake frame count to 0
fake_detected = False  # Initialize fake_detected flag to False

for frameId in frameNumbers:
    cap.set(cv2.CAP_PROP_POS_FRAMES, frameId)
    ret, frame = cap.read()
    if ret != True:
        break
    face_rects, _, _ = detector.run(frame, 0)
    if len(face_rects) == 0:  # Check if no faces were detected
        continue  # Skip this frame
    for d in face_rects:
        x1, y1, x2, y2 = d.left(), d.top(), d.right(), d.bottom()
        crop_img = frame[y1:y2, x1:x2]
        data = img_to_array(cv2.resize(crop_img, (128, 128))).flatten() / 255.0
        data = data.reshape(-1, 128, 128, 3)
        score = model.predict(data)[0][1] # probability of being fake
        if score == 1.0:  # Check if score equals 1.0 (100%)
            fake_detected = True
        if score > 0.5:  # Check if score exceeds 50%
            fake_frame_count += 1
        # Draw a rectangle around the detected face
        cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
        # Add a label with the probability of fakeness to the top of the face frame
        cv2.putText(frame, f'Fakeness: {score:.2f}', (x1, y1-10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
    # Display the video frame with the detected faces and probability of fakeness
    plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
    plt.show()

if fake_detected:  # Check if any face has 100% probability of being fake
    print("The video is classified as FAKE.")
elif fake_frame_count >= 3 and fake_frame_count <= 5:  # Check if fake was detected in
3-5 frames
    print("The video is classified as POSSIBLY FAKE.")
elif fake_frame_count > 5:  # Check if fake was detected in more than 5 frames
    print("The video is classified as FAKE.")
else:  # Otherwise, the video is classified as real
    print("The video is classified as REAL.")

import numpy as np
import random
import matplotlib.pyplot as plt
```

```
input_shape = (128, 128, 3)
pr_data = []
detector = dlib.get_frontal_face_detector()
cap = cv2.VideoCapture('/content/sample_data/CCqaaV-vtmtmDifI.mp4')
frameCount = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
frameNumbers = random.sample(range(frameCount), 20) # Select 20 random frame
numbers
fake_frame_count = 0  # Initialize fake frame count to 0
fake_detected = False  # Initialize fake_detected flag to False

for frameId in frameNumbers:
    cap.set(cv2.CAP_PROP_POS_FRAMES, frameId)
    ret, frame = cap.read()
    if ret != True:
        break
    face_rects, _, _ = detector.run(frame, 0)
    if len(face_rects) == 0:  # Check if no faces were detected
        continue  # Skip this frame
    for d in face_rects:
        x1, y1, x2, y2 = d.left(), d.top(), d.right(), d.bottom()
        crop_img = frame[y1:y2, x1:x2]
        data = img_to_array(cv2.resize(crop_img, (128, 128))).flatten() / 255.0
        data = data.reshape(-1, 128, 128, 3)
        score = model.predict(data)[0][1] # probability of being fake
        if score == 1.0:  # Check if score equals 1.0 (100%)
            fake_detected = True
        if score > 0.5:  # Check if score exceeds 50%
            fake_frame_count += 1
        # Draw a rectangle around the detected face
        cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
        # Add a label with the probability of fakeness to the top of the face frame
        cv2.putText(frame, f'Fakeness: {score:.2f}', (x1, y1-10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
    # Display the video frame with the detected faces and probability of fakeness
    plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
    plt.show()

if fake_detected:  # Check if any face has 100% probability of being fake
    print("The video is classified as FAKE.")
elif fake_frame_count >= 3 and fake_frame_count <= 5:  # Check if fake was detected in
3-5 frames
    print("The video is classified as POSSIBLY FAKE.")
elif fake_frame_count > 5:  # Check if fake was detected in more than 5 frames
    print("The video is classified as FAKE.")
else:  # Otherwise, the video is classified as real
    print("The video is classified as REAL.")
```

B. SNAPSHOTS

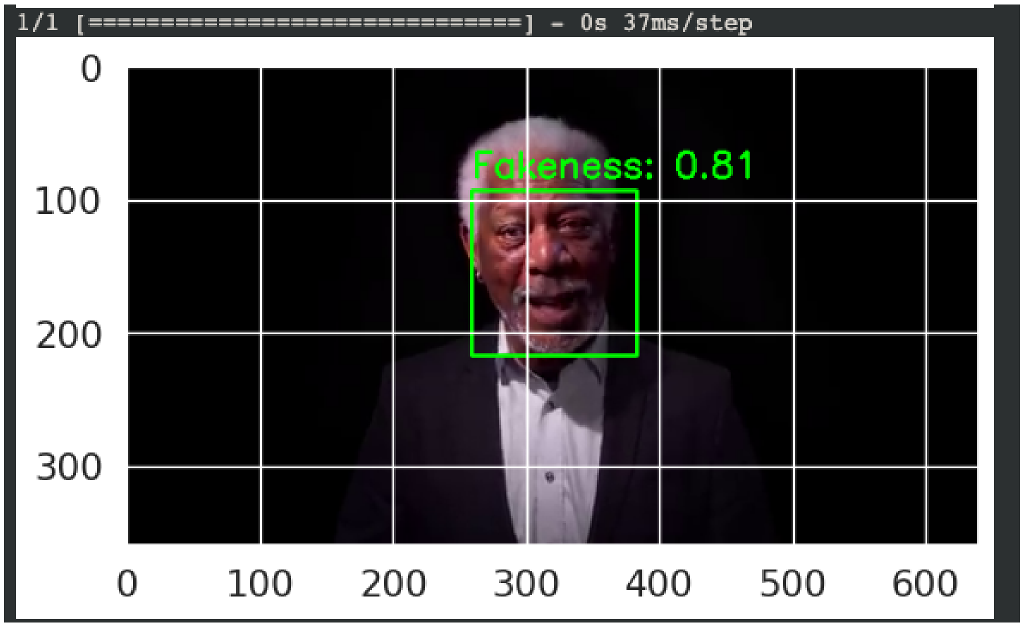






```
1/1 [==============================] - 0s 56ms/step
```

```
The percentage of fakeness is: 2.945440635085106
The maximum probability of fakeness is: 0.13899852
The minimum probability of fakeness is: 0.00012755652
```



```
1/1 [==============================] - 0s 37ms/step
```

REFERENCES

[1] Rafael C. Gonzalez and Richard E. Woods, Digital Image Processing, Third Ed., Prentice Hall, 2008.

[2] Darius Afchar, Vincent Nozick, Junichi Yamagishi, Isao Echizen: MesoNet: a Compact Facial Video Forgery Detection Network, 2018

[3] Yuezun Li, Ming-Ching Chang and Siwei Lyu: In Ictu Oculi: Exposing AI Generated Fake Face Videos by Detecting Eye Blinking, 2018.

[4] Yuezun Li, Siwei Lyu: Exposing DeepFake Videos By Detecting Face Warping Artifacts, 2019.

[5] S. Sridhar, Digital Image Processing, Oxford University Press, 2ndEdition, 2016.

[6] Digital Image Processing- S.Jayaraman, S.Esakkirajan, T.Veerakumar, TataMcGraw Hill 2014.

[7] Fundamentals of Digital Image Processing-A. K. Jain, Pearson 2004.

[8] https://www.kaggle.com/competitions/deepfake-detection-challenge/data