Classification & Recognition of Real and Fake Human Faces Using Deep Learning

DEPARTMENT OF COMPUTER SCIENCE & SYSTEMS ENGINEERING

ANDHRA UNIVERSITY COLLEGE OF ENGINEERING

ANDHRA UNIVERSITY

VISAKHAPATNAM

CERTIFICATE

This is to certify that this is a bonafide project work entitled **"Classification & Recognition of Real and Fake Human Faces Using Deep Learning"**, doneby **Mr. BONGU SAI KUMAR (Reg. No: 319506402013)** and **Mr. BOOSA ABISHEK REDDY (Reg. No: 319506402014)** and **Mr. BORA SANTHOSH (Reg. No: 319506402015)**, students of Department of Computer Science & Systems Engineering, Andhra University College of Engineering (A) during the period 2019-2023 in the partial fulfillment of the requirements for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering**.

This report has not been submitted to any other college or university.

Prof.B. Prajna	**Prof. K. Venkata Rao**
Project Guide	Head of the Department
Dept of CS&SE	Dept of CS&SE
AUCE	AUCE
Visakhapatnam	Visakhapatnam

DEPARTMENT OF COMPUTER SCIENCE & SYSTEMS ENGINEERING

ANDHRA UNIVERSITY COLLEGE OF ENGINEERING

ANDHRA UNIVERSITY

VISAKHAPATNAM

DECLARATION

We hereby declare that the project entitled **"Classification & Recognition of Real and Fake Human Faces Using Deep Learning"** has been prepared by us during the period November 2022 – April 2023 in partial fulfillment of the requirements for the award of degree of **Bachelor of Technology** in **Computer Science and Systems Engineering.**

BONGU SAI KUMAR

(Reg. No:319506402013)

BOOSA ABISHEK REDDY

(Reg. No: 319506402014)

BORA SANTHOSH

(Reg. No: 319506402015)

Place: Visakhapatnam

Date:

ACKNOWLEDGMENT

We have immense pleasure in expressing our earnest gratitude to our Project Guide **Prof. B.Prajna**, Andhra University for her inspiring and scholarly guidance. Despite her preoccupation with several assignments, she has been kind enough to spare her valuable time and gave us the necessary counsel and guidance at every stage of planning and constitution ofthis work. We express sincere gratitude for having accorded us permission to take up this project work and for helping us graciously throughout the execution of this work.

We express sincere Thanks to **Prof. K. Venkata Rao**, Head of the Department Computer Science and Systems Engineering, Andhra University College of Engineering (A) for his keen interest and providing necessary facilities for this project study**.**

We express sincere gratitude to **Prof. P.V.G.D. Prasad Reddy**, Vice Chancellor , Andhra University College of Engineering(A) for his keen interest and providing necessary facilities for this project study.

We express sincere Thanks **to Prof. G. Sasibushana Rao**, Principle, Computer Science and Systems Engineering, Andhra University College of Engineering(A) for his keeninterest and providing necessary facilities for this project study.

We extend our sincere thanks to our academic teaching staff and non-teaching staff for their help throughout our study.

ABSTRACT

Artificial intelligence (AI), deep learning, machine learning and neural networks represent extremely exciting and powerful machine learning-based techniques used to solve many real-world problems. Artificial intelligence is the branch of computer sciences that emphasizes the development of intelligent machines, thinking and working like humans. For example, recognition, problem-solving, learning, visual perception, decision-making and planning. Deep learning is a subset of machine learning in artificial intelligence that has networks capable of learning unsupervised from data that is unstructured or unlabeled.

Deep learning is a technique used to generate face detection and recognize it for real or fake by using profile images and determine the differences between them. In this study, we used deep learning techniques to generate models for Real and Fake face detection.

The goal is determining a suitable way to detect real and fake faces. The model was designed and implemented, including both Dataset of images: Real and Fake faces detection through the use of Deep learning algorithms based on neural networks. We have trained dataset which consists of 5171 images for total in 25 epochs, and got the ImageNet model to be the best model of network architectures used with 93% training accuracy, 91.45% validation accuracy, training loss 0.0003, validation loss 0.0265.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS/NOTATIONS

IP-Image processing

CNN-Convolutional neural network

Haar-Haar cascades

UML-Unified modelling language

Opnecv-Open Source ComputerVision Library

Relu-Rectified Linear Activation Function

CHAPTER 1

INTRODUCTION

Machine learning is an application of artificial intelligence that uses statistical techniques to enable computers to learn and make decisions without being explicitly programmed. It is predicated on the notion that computers can learn from data, spot patterns, and make judgments with little assistance from humans.

Terminology:

- **Model**: Also known as "hypothesis", a machine learning model is the mathematical representation of a real-world process. A machine learning algorithm along with the training data builds a machine learning model.
- **Feature**: A feature is a measurable property or parameter of the data-set.
- **Feature Vector**: It is a set of multiple numeric features. We use it as an input to the machine learning model for training and prediction purposes.
- **Training**: An algorithm takes a set of data known as "training data" as input. The learning algorithm finds patterns in the input data and trains the model for expected results (target). The output of the training process is the machine learning model.
- **Prediction**: Once the machine learning model is ready, it can be fed with input data to provide a predicted output.
- **Target (Label)**: The value that the machine learning model has to predict is called the target or label.
- **Overfitting**: When a massive amount of data trains a machine learning model, it tends to learn from the noise and inaccurate data entries. Here the model fails to characterize the data correctly.

- **Underfitting**: It is the scenario when the model fails to decipher the underlying trend in the input data. It destroys the accuracy of the machine learning model. In simple terms, the model or the algorithm does not fit the data well enough

There are Seven Steps of Machine Learning

1. Gathering Data
2. Preparing that data
3. Choosing a model
4. Training
5. Evaluation
6. Hyperparameter Tuning
7. Prediction

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves. The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide.

The primary aim is to allow the computers learn automatically without human intervention or assistance an adjust actions accordingly. Deep learning is a subset of machine learning in artificial intelligence that has networks capable of learning unsupervised from data that is unstructured or unlabeled. Deep learning is a technique used to generate face detection and recognize it for real or fake by using profile images and determine the differences between them. In this study, we will

be using deep learning techniques to generate model for Real and Fake face detection. The goal is determining a suitable way to detect real and fake faces.

Fake and Real Human Face

With the development of Generative Adversarial Network (GAN) computers can generate vivid face images that can easily deceive human beings. These generated fake faces will inevitably bring serious social risks, e.g., fake news, fake evidence, and pose threats to security .

Thus, powerful techniques to detect these fake faces are highly desirable. However, in contrast to the intensive studies in GANs, our understanding of generated faces is fairly superficial and how to detect fake faces is still an under-explored problem.

Moreover, fake faces in practical scenarios are from different unknown sources, i.e. different GANs, and may undergo unknown image distortions such as down sampling, blur, noise, and JPEG compression, which makes this task even more challenging. In this study, we aim to produce new insights on understanding fake faces and propose a new architecture to tackle these challenges.

A facial spoof attack is a process in which a fraudulent user attacks the face recognition system by disguising as an original or authorized user and thereby gaining unauthorized access and benefits. The face spoofing attack can be done by the unauthorized user by collecting the original user's data such as photographs, videos through social media or by capturing photos on Mobile. The fake evidence can be used by an illegal user to gain access. The face spoofing attacks are categorized in 2D and 3D spoof attacks. The 2D spoof consists of a photo and Video attack and in a 3D spoof.

Convolutional Neural Network (CNN)

Neural network with a convolution operation instead of matrix multiplication in at least one of the layers. It's one of the main categories to do images recognition, images classifications. Technically, deep learning CNN models to train and test, each input image will pass it through a series of convolution layers with filters (Kernels), polling, fully connected layers (FC) and apply SoftMax function to

classify an object with probabilistic values between 0 and 1. The below figure is a complete flow of CNN to process an input image and classifies the objects based on values .

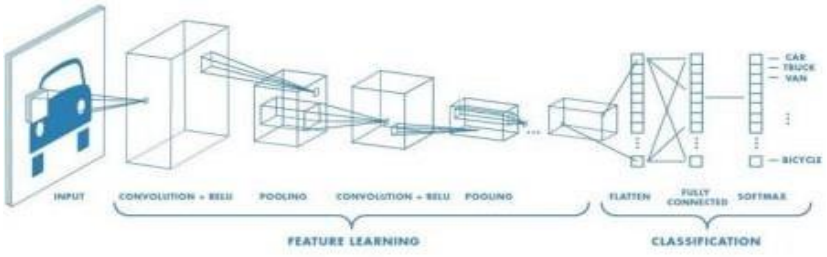

Figure 1: Convolutional Neural Network

Rectified Linear Unit Layer (ReLU)

It is a type of activation function. Mathematically, it's defined as y = max (0, x). ReLU is the most commonly used function in neural networks, especially in CNNs. ReLU is linear for all positive values, and zero for all negative values. It's cheap to compute as there is no complicated math. The model can therefore take less time to train or run. It converges faster and sparsely activated. Since ReLU is zero for all negative inputs, it's likely for any given unit to not activate at all. The advantage of not having any backpropagation errors, also for larger neural networks, the speed of building models based of on ReLU is very fast. ReLU isn't without any drawbacks some of them are that ReLU is nonzero centered and is non-differentiable at zero, but differentiable anywhere else .

PROBLEM STATEMENT

According to use of social media, we will encounter a fake identity from any person through using fake profile image. The fake profile image can be occurred with using image editor, face effect, or any program using to change the facial features.

Effects on the face can change facial features and it's difficult to know true identity for someone. In this application, we hope to train a model using the dataset to recognize such fake faces.

Computerized applications can use deep learning techniques to increase accuracy and efficiency in diagnosis. These include human images, image processing techniques and data analysis.

MOTIVATION

The motivation for face antispoofing detection using deep learning is to develop reliable and robust methods that can detect and prevent spoof attacks in real-world scenarios. Deep learning models have shown promising results in learning complex and discriminative features from face images, which can distinguish between genuine and fake faces.

Moreover, deep learning-based face antispoofing methods can be trained on large datasets with diverse and challenging spoofing scenarios, which can improve their generalization and adaptability to various environments and applications. This can enhance the security and reliability of facial recognition systems and prevent potential privacy and identity theft issues.

Objectives

Main objective for this study is implementation a software model used to detect and classify face to real or fake for expertgenerated high-quality photo shopped face images.

Other objectives are:

- Detect fake face images rapidly.
- Get high accuracy and validation in the testing and training images.
- Reduce cost and effort of diagnosis and repetitive images.
- Implementation network architectural to find the best model with perfect result.
- Training high dataset of fake and real human face to see an accuracy of the result

CHAPTER 2

LITERATURE SURVEY

Many research papers have been published to use artificial intelligence, expert systems, and neural networks to improve the detection of fake face. Recently, social media have begun to introduce artificial intelligence systems and applications in some disciplines to increase the accuracy of profile images detection. Many methods and models have been introduced that have contributed to increased detection of the fake face in an efficiency. In one of paper, the researcher has been evaluated the generalizability of the fake face detection methods through a series of studies to benchmark the detection accuracy. To this extent, they have collected a new database of more than 53,000 images, from 150 videos, originating from multiple sources of digitally generated fakes including Computer Graphics Image (CGI) generation and many tempering-based approaches. Also, they have included images (with more than 3,200) from the predominantly used SwapFace application.

Extensive experiments are carried out using both texture-based handcrafted detection methods and deep learningbased detection methods to find the suitability of detection methods . Comment on the previous study: the difference are they have aim measurement the evaluation of generalizability on exising fake face detection techniques. So they focused to create a new database which they refer to as Fake Face in the Wild (FFW) dataset by using texture-based methods – Local Binary Patterns (LBP) with support Vector Machine (SVM) and a set of CNN architectures like ImageNet, keras ,etc..

In other paper, the researcher has been proposed neural network-based classifier to detect fake human faces created by machines and humans, so they have been using ensemble methods to detect Generative Adversarial Networks (GANS) created fake images and employ pre-processing techniques to improve fake face image detection created by humans. In this study, the researcher has been focused on image contents for classification and do not use meta-data of images.

The result can be effectively detected both GANS-created images and human-created fake images with 94% and 74.9% AUROC (Area Under Receiver Operating

Characteristic) Score . Comment on the previous study: we difference from this study by using large dataset of real and fake image has several changes of facial features and training it more of model in Network Architectures and get high percentage of accuracy and validation against this study.

Another paper, the researcher has been proposed a multi-feature fusion scheme that combines dynamic and static joint analysis to detect fake face attacks. According to can be easily detected the texture differences between the real and the fake faces, Local Binary Pattern (LBP) texture operators and optical flow algorithms are often merged. Also, in the traditional optical flow algorithm is also modified by applying the multi-faction feature super-position method, which reduces the noise of the image. In the pyramid model, image processing is performed in each layer by using block calculations that form multiple block images.

The features of the image are obtained via two fused algorithms (MOLF), which are then trained and tested separately by an SVM classifier. Experimental results show that this method can improve detection accuracy while also reducing computational complexity. In this paper, the researcher has been used the CASIA, PRINT-ATTACK, and REPLAY-ATTACK database to compare the various LBP (Local Binary Patter) algorithms that incorporate optical flow and fusion algorithms . Comment on the previous study: this study focuses on photo attack, video attack, image with real eye attack, and the total three-way attack, and it is using difference model to detect the fake face attack and get the high accuracy 96.543%, also using MATLAB to implementation the result.

CHAPTER 3

REQUIREMENT ANALYSIS

Requirement analysis for face antispoofing detection system based on the ImageNet dataset and Keras, and trained using the MobileNet architecture, can be broken down into several categories:

Functional Requirements:

The system should be able to capture input images from various sources.

The system should detect a face from the input image.

The system should apply the anti-spoofing model to the detected face.

The system should predict whether the detected face is genuine or spoofed.

The system should display the results of the prediction to the user.

Non-Functional Requirements:

The system should have a low false-positive rate, i.e., it should accurately detect spoofed faces and not classify genuine faces as spoofed.

The system should have a low false-negative rate, i.e., it should accurately detect genuine faces and not classify them as spoofed.

The system should be able to process images in real-time, i.e., within a reasonable time frame.

The system should have high accuracy and robustness to different types of spoof attacks, including printed photos, digital screens, and 3D masks.

The system should be user-friendly and easy to operate, even for non-experts.

Technical Requirements:

The system should be implemented using Keras and the MobileNet architecture.

The system should be trained using a large and diverse dataset, such as the ImageNet dataset.

The anti-spoofing model should be optimized using appropriate loss functions and regularization techniques.

The system should be tested and evaluated using standard metrics, such as the Area Under the Curve (AUC) and Equal Error Rate (EER).

Security and Privacy Requirements:

The system should be secure and prevent unauthorized access to sensitive information.

The system should not store or transmit any personal or sensitive data, such as biometric information, without the user's consent.

The system should comply with relevant privacy laws and regulations, such as the General Data Protection Regulation (GDPR) and the California Consumer Privacy Act (CCPA).

Deployment and Maintenance Requirements:

The system should be easy to deploy and integrate with other applications and systems.

The system should be scalable and able to handle large volumes of images and requests.

The system should be maintainable and supportable, with appropriate documentation and user manuals.

The system should receive regular updates and patches to address any security or performance issues.

SOFTWARE INSTALLATIONS REQUIREMENTS(Requirements.txt)

Python 3.6 &

```
pencv-python==4.4.0.40
tensorflow==2.5.0
opencv-python
cmake==3.17.2
dlib==19.18.0
Pillow
face_recognition
geopy
requests
easydict==1.10
numpy
tqdm
torchvision
torch
```

CHAPTER 4

SYSTEM ARCHITECTURE

The researcher has used deep learning and transfer learning approach to build the face anti-spoofing module to detect the face spoofing attack. The pre-trained model MOBILENET is used that is already trained on a large ImageNet dataset. The model is customized by modifying the last layers and again re-trained on the dataset . The trained model is used to solve the binary classification problem to classify or predict the fake and real class of users. To detect the face from the image we have used the harrcascadeforntalfacem algorithm and to extract the feature embeddings the keras model is used from the transfer learning . To build the model Keras library and tensorFlow are used:

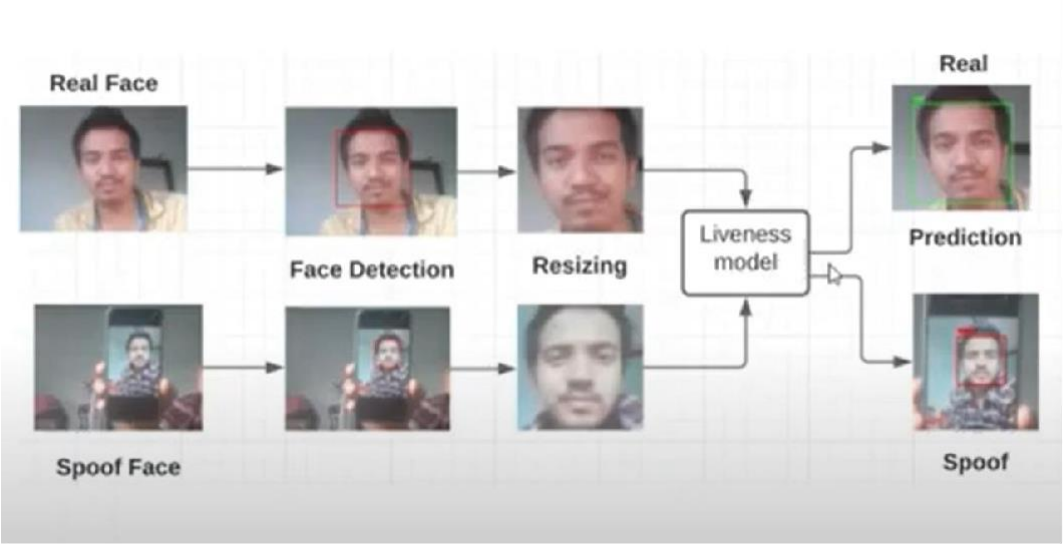

FIG.2 ARCHITECTURE

DESIGN

These UML diagrams show the basic structure and flow of the face antispoofing detection system based on the ImageNet dataset and Keras, and trained using the MobileNet architecture. The system first detects a face using a face detector, then applies an anti-spoofing model to the detected face to predict whether it is genuine or spoofed. The system then displays the results of the prediction to the user.

1. USE CASE DIAGRAM

FIG 3 USE CASE DIAGRAM

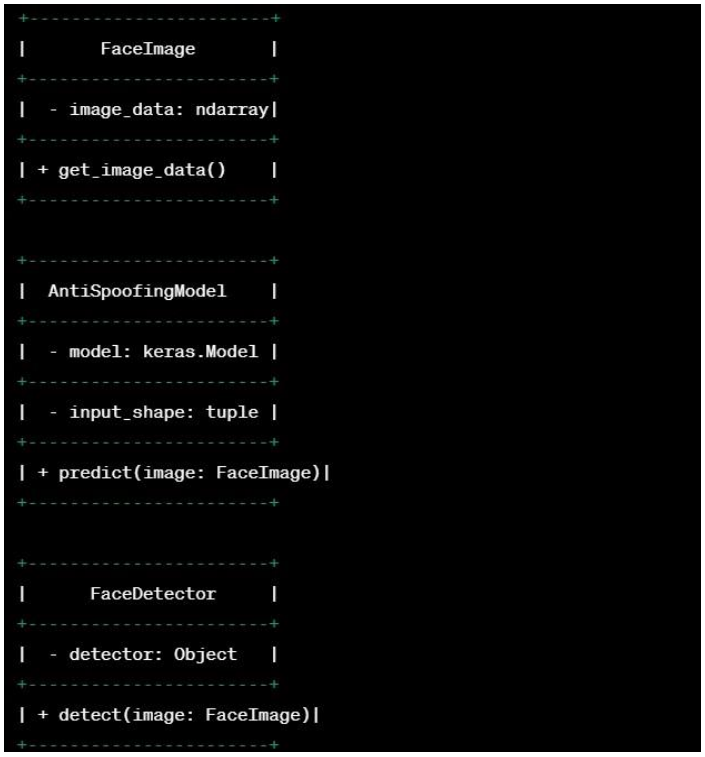

FIG 4 CLASS DIAGRAM

1. 

Fig 5 ACTIVITY DIAGRAM

METHODOLOGY

1. Initially, the dataset is loaded and different data pre-processing steps are applied on the dataset.

2. After applying the pre-processing steps, the data split into a training set and testing set in the 80:20 ratio respectively.

3. The transfer learning MOBILENET model is created by customizing the last dense layer and the two classes are added fake and real at the output layer.

4. The model is trained on the dataset and then evaluated for training accuracy.

5. The trained model again validated on test dataset to evaluate the validation accuracy of the model.

6. The developed model is saved and loaded to evaluate the fake and real class of users**.**

CHAPTER 5

IMPLEMENTATION AND TESTING

Implementation of dataset

The data set in this study consists of 5171 human faces of real-fake images. The numbers of images in this dataset are classified as follows: 2489 images that were real human faces, 2592 of fake. Then the dataset collection has been categorized into four types, uploaded to a Google Drive account and verified to be properly and accurately uploaded using Python code in the Google Colab environment.

DISTRIBUTION OF IMAGES IN THE HOME DIRECTORY

	TOTAL IMAGES	TRAINING	TESTING
REAL IMAGES	2489	2102	477
FAKE IMAGES	2592	2118	474
TOTAL	5171	4130	951

TABLE 1

Image format

Dataset was collected from a set of real-fake human faces Images for detecting weather an image is real or fake (JPG) format, in order to fit well with the model used to give the desired results.

Data augmentation

Generating more data usually means that the model will be more robust and prevent overfitting. Having a large dataset is crucial for the performance of the deep learning model. However, I improved the performance of the model by augmenting the images that I already have without collecting new images. Deep learning frameworks usually have built-in library for data augmentation utilities; I utilized five augmentation strategies to generate new training sets, (Rotation, width shift,

height shift, horizontal flip, and vertical flip) . Rotation augmentations are done by rotating the image right or left on an axis between 1° and 360°. The safety of rotation augmentations is heavily determined by the rotation degree parameter. Shifting and flipping images are a very useful transformation to encapsulating more details about objects of interest.

5.1 Getting datasets from google drive

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

This code snippet mounts your Google Drive to your Colab notebook's file system. After running this code, you will be prompted to click on a link, sign in with your Google account, and copy a verification code. After entering the code in the text box that appears in the output of the cell, your Google Drive will be mounted and you can access your files stored in Drive through the /content/gdrive/ directory in the Colab notebook.

Note that this code snippet is specific to the Colab environment and will not work in other Python environments.

```
!cp -r "/content/gdrive/MyDrive/final_antispoofing.zip" "/content"
```

This code snippet copies a file named final_antispoofing.zip from your Google Drive to the root directory of your Colab notebook's file system. The !cp command stands for "copy" and the -r flag indicates that the copy should be recursive, meaning it will copy all files and subdirectories within the specified directory.

After running this code, you should be able to see the final_antispoofing.zip file in the root directory of your Colab notebook's file system.

```
import zipfile
```

```
archive = zipfile.ZipFile("/content/final_antispoofing.zip")
archive.extractall('/content')
```

This code snippet extracts all the contents of the final_antispoofing.zip file in the /content directory of your Colab notebook's file system.

The zipfile.ZipFile() function is used to create a ZipFile object that represents the archive file. The file path /content/final_antispoofing.zip specifies the location of the archive file in your Colab notebook's file system.

The extractall() method of the ZipFile object is then used to extract all the contents of the archive file to the /content directory. The argument /content specifies the location where the contents should be extracted to.

After running this code, all the files in the final_antispoofing.zip archive should be extracted and available in the /content directory of your Colab notebook's file system.

DATA PREPROCESSING

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model.

When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So for this, we use data preprocessing task.

NEED FOR DATA PREPROCESSING

A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data preprocessing is required tasks for cleaning the data and making it suitable for a

machine learning model which also increases the accuracy and efficiency of a machine learning model.

It involves below steps:

- Getting the dataset
- Importing libraries
- Importing datasets
- Finding Missing Data
- Encoding Categorical Data
- Splitting dataset into training and test set
- Feature scaling

We have included these four steps in our project, they are

Step 1: Create new directory structure for the datasets

Step 2: Copying images into new directory structure

Step 3: Dataset Exploration

Step 4: Dataset Visualization

STEP 1 – ORIGINAL vs NEW DIRECTORY

```
from IPython import display
print("Original Vs New Dataset Structure")
display.Image('original vs new_dataset.png')
```

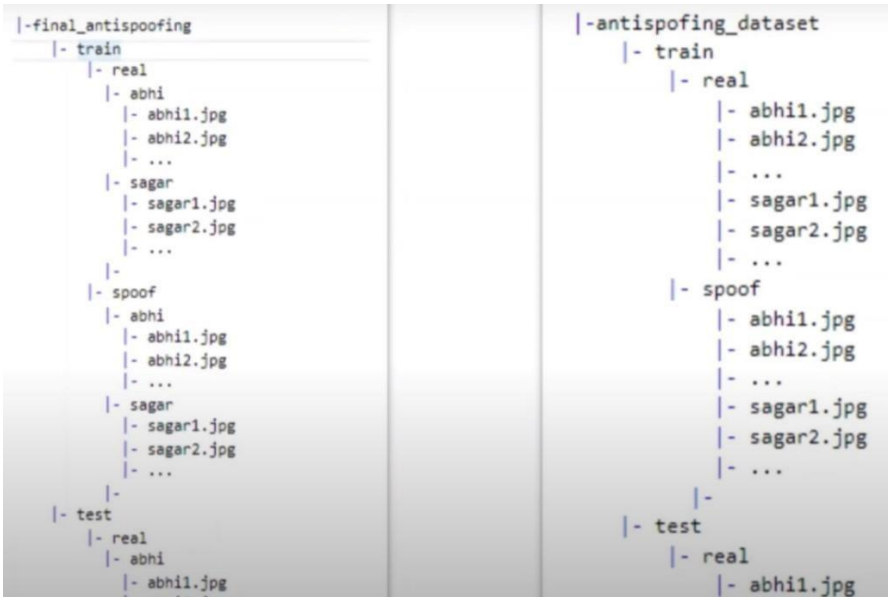

FIG 6 DIRECTORY VISUALISATION

```
dataset_dir = '/content/final_antispoofing'
train_dataset_dir = '/content/final_antispoofing/train'
test_dataset_dir = '/content/final_antispoofing/test'
```

dataset_dir is the root directory of the dataset, train_dataset_dir is the directory containing the training data, and test_dataset_dir is the directory containing the testing data.

```
import  os
os.mkdir('/content/antispoofing_dataset')
os.mkdir('/content/antispoofing_dataset/train')
os.mkdir('/content/antispoofing_dataset/test')
os.mkdir('/content/antispoofing_dataset/train/real')
os.mkdir('/content/antispoofing_dataset/train/spoof')
os.mkdir('/content/antispoofing_dataset/test/real')
os.mkdir('/content/antispoofing_dataset/test/spoof')
```

The os.mkdir() function is used to create the directories.

```
train_dir='/content/antispoofing_dataset/train'
test_dir = '/content/antispoofing_dataset/test'
```

These code lines define two variables, train_dir and test_dir, which contain the file paths to the directories of the preprocessed training and testing data in your Colab notebook's file system, respectively.

STEP 2- COPYING IMAGES TO NEW DIRECTORY

```
import  shutil
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

shutil is a module in Python's standard library that provides high-level file operations such as copying, moving, and deleting files and directories.
numpy is a library for numerical computations in Python, which provides support for multi-dimensional arrays and matrices.
matplotlib is a plotting library in Python which provides a wide range of visualization tools.
cv2 is the OpenCV library for computer vision tasks in Python.

```
def train_test_splits(data_directory):
  for split_type in os.listdir(data_directory):
    path_to_split_type = os.path.join(data_directory,split_type)
    for category in os.listdir(path_to_split_type):
      path_to_category = os.path.join(path_to_split_type,category)
      for subject in os.listdir(path_to_category):
        path_to_subject = os.path.join(path_to_category,subject)
        for img in os.listdir(path_to_subject):
          if split_type == 'train':
```

```
            shutil.copy(os.path.join(path_to_subject,img),os.path.join(train_dir,catego
ry,img))
        else:
            shutil.copy(os.path.join(path_to_subject,img),os.path.join(test_dir,categor
y,img))

train_test_splits(data_directory=dataset_dir)
```

Overall, this function performs the train-test split on the preprocessed dataset and copies the images to the corresponding train and test directories for further processing.

STEP 3- DATASET EXPLORATION

```
categories = ['real','spoof']
print("----------------------Exploring Training Datasets ------------------- ")
for category in categories:
  path = os.path.join(train_dir,category)
  if category == 'real':
    r1 = len(os.listdir(path))
  else:
    s1 = len(os.listdir(path))
  print("There are {} images in {} directory".format(len(os.listdir(path)),category)
)
print("There are {} total images in training directory".format(r1+s1))

print("..... Exploring Testing Datasets ..... ")
for category in categories:
  path = os.path.join(test_dir,category)
  if category == 'real':
    r2 = len(os.listdir(path))
  else:
    s2 = len(os.listdir(path))
```

```
  print("There are {} images in {} directory".format(len(os.listdir(path)),category)
)
print("There are {} total images in testing directory".format(r2+s2))
```

```
------------------------------Exploring Training Datasets----------------------
--------
There are 2102 images in real directory
There are 2118 images in spoof directory
There are 4220 total images in training directory
-------------------------------Exploring Testing Datasets-----------------------
---------------
There are 477 images in real directory
There are 474 images in spoof directory
There are 951 total images in testing directory
```

STEP 4 : DATASET VISUALISATION

```
def get_images(data_dir,number_of_samples):
  image_path = []
  for category in categories:
    path = os.path.join(data_dir,category)
    i = 1
    for img in os.listdir(path):
      if i > number_of_samples:
        break
      else:
        image_path.append(os.path.join(path,img))
        i += 1
  return image_path
```

the function returns the image_path list containing the file paths of the selected images.

```
def visualize_dataset(image_path,rows,cols):
```

```
  fig = plt.figure(figsize=(20,20))
  for i in range(1,rows * cols + 1):
  fig.add_subplot(rows,cols,i)
    img_array = cv2.imread(image_path[i-1])
    fig.subplots_adjust(hspace=1)
    plt.imshow(cv2.cvtColor(img_array,cv2.COLOR_BGR2RGB))
    plt.xlabel(image_path[i-1].split('/')[-2])
  plt.show()
```

This function takes in a list of image paths (image_path) and plots a grid of images with the specified number of rows and cols, where each image is labeled with its category (obtained from the parent folder name of the image path).

```
training_image_path = get_images(data_dir= train_dir,number_of_samples=25)
print(training_image_path)
print(len(training_image_path))
testing_image_path = get_images(data_dir= test_dir,number_of_samples=25)
print(testing_image_path)
print(len(testing_image_path))
```

This code calls the get_images() function to retrieve the file paths for 25 samples of each category (real and spoof) in the training dataset.

```
visualize_dataset(image_path=training_image_path,rows=5,cols=10)
```

This code calls the visualize_dataset() function to display a grid of images.

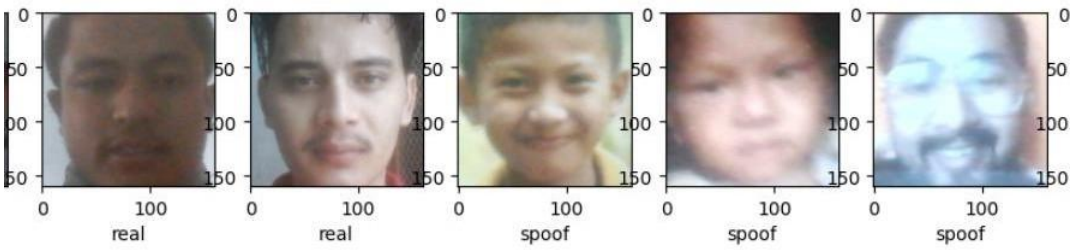

FIG 7 TRAIN DATASET VISUALISATION

visualize_dataset(image_path=testing_image_path,rows=5,cols=10)

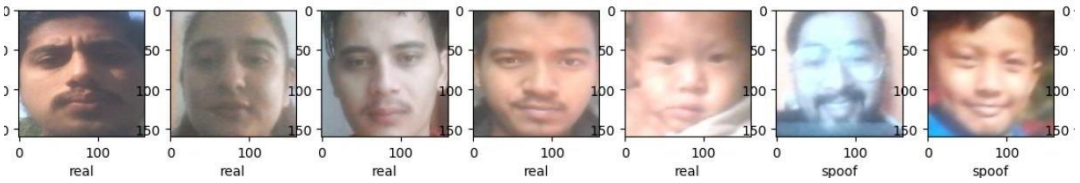

FIG 8 TEST DATASET VISUALISATION

ABOUT KERAS

Keras is a high-level neural networks API, written in Python and capable of running on top of various lower-level machine learning libraries such as TensorFlow, Theano, and CNTK. It provides a user-friendly interface for building deep learning models with a few lines of code.

Some of the features of Keras include:

User-friendly interface: Keras provides a simple and intuitive interface for building deep learning models, making it easier for developers and researchers to create complex models with fewer lines of code.

Modular and extensible: Keras is modular, meaning that you can build deep learning models by assembling pre-built building blocks called "layers". Keras provides a large number of layers to choose from, including convolutional layers, recurrent layers, pooling layers, and more. It is also extensible, allowing you to define your own layers and models.

GPU support: Keras supports running deep learning models on GPUs, which can significantly speed up training times for large models.

Visualization tools: Keras provides tools for visualizing the structure of your models, including the ability to plot the architecture of a model as a graph.

Preprocessing and data augmentation: Keras provides tools for preprocessing data and performing data augmentation, such as image rotation and cropping, which can help improve the performance of your models.

Easy deployment: Keras models can be easily deployed to a variety of platforms, including mobile devices and the web.

Overall, Keras is a powerful and flexible deep learning framework that is widely used by researchers and developers for a variety of applications, including computer vision, natural language processing, and speech recognition

Model Preparation

Steps In Model Preparation

1. Choosing Framework and importing necessary libraries

2. Load datasets and Perform image augmentations

3. Model Selection

4. Compiling our model

5. Setting our model checkpoints

```
from keras.layers import Dense,Dropout,Input,Flatten

from keras.models import Model

from keras.optimizers import Adam

from keras.preprocessing.image import ImageDataGenerator

from keras.callbacks import ModelCheckpoint

from keras.applications.mobilenet_v2 import MobileNetV2

from keras.models import model_from_json

import json

train_datagen =
ImageDataGenerator(brightness_range=(0.8,1.2),rotation_range=30,width_shift_r
```

```
ange=0.2,height_shift_range=0.2,fill_mode='nearest',shear_range=0.2,zoom_rang
e=0.3,rescale=1./255)

valid_datagen = ImageDataGenerator(rescale=1./255)

train_generator =
train_datagen.flow_from_directory(train_dir,target_size=(160,160),color_mode='r
gb',  class_mode='binary',batch_size=25,shuffle=True)


valid_generator =
valid_datagen.flow_from_directory(test_dir,target_size=(160,160),color_mode='r
gb',class_mode='binary',batch_size=25)

mobilenet =
MobileNetV2(weights="imagenet",include_top=False,input_tensor=Input(shape=
(160,160,3)))

mobilenet.trainable = False

output = Flatten()(mobilenet.output)

output = Dropout(0.3)(output)

output = Dense(units = 8,activation='relu')(output)

prediction = Dense(1,activation='sigmoid')(output)

model = Model(inputs = mobilenet.input,outputs = prediction)

model.summary()

# tell the model what cost and optimization method to use

model.compile(

  loss='binary_crossentropy',

  optimizer=Adam(
```

```
    learning_rate=0.000001,
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1e-07
),metrics=['accuracy'])


import os
os.mkdir('/content/model_weights/')
model_checkpoint = ModelCheckpoint('./model_weights/finalyearproject_antispoofing_model_{epoch:02d}-{val_accuracy:.6f}.h5', monitor='val_loss', mode='min', verbose=1, save_best_only=True,save_weights_only=True)
history = model.fit_generator( train_generator, steps_per_epoch = train_generator.samples // 25, validation_data = valid_generator, validation_steps = valid_generator.samples // 25,
  epochs = 25, callbacks=[model_checkpoint])
```

EXPLAINATION FOR ABOVE CODE

The code imports necessary modules from Keras, sets up data generators, loads a pre-trained MobileNetV2 model with frozen weights, adds layers to the model, compiles the model, creates a directory to save model weights, sets up a model checkpoint to save the best performing model during training, and fits the model to the training data using the generators and validates it using the validation generator for 25 epochs. The final trained model is saved to the specified directory with its accuracy and epoch number in the filename.

Network Architecture

We have trained our Real-Fake dataset using a model that we created from scratch FOR deep learning: MobileNet

Training and Validating the Models

During the model training, the model will iterate over batches of the training dataset, each of size batch_size. For each batch, gradients will be computed, and updates will be made to the weights of the network automatically. One iteration over all of the training dataset is referred to as an epoch. Training is usually run until the loss converges to a small constant.

We used checkpoint in the model to save the best validation accuracy. This is useful because the network might start overfitting after a certain number of epochs. This feature was implemented via the callback feature of the library Keras. Callback is a set of functions that is applied at given stages of training procedure like end of an epoch of training. Keras provides built-in function

for both learning rate scheduling and model check-pointing.

MOBILENET MODEL

I used a pre-trained model called MobileNet network with convolutional layers followed by a fully connected hidden layer, also used dropout layers in between, dropout regularizes the networks to prevent the network from overfitting, all layers have ReLU activation function. We trained the Real-Fake dataset using the pre-trained MobileNet model and the training accuracy reached 95.84% and validation accuracy reached 97.37%. below Figures illustrates the accuracy of both training and validation of MobileNet model of the Real-Fake dataset.

MobileNetV2 is a pre-trained convolutional neural network (CNN) architecture designed for mobile devices. It is a smaller and faster version of its predecessor, MobileNetV1, which makes it more suitable for resource-constrained environments such as mobile devices. The architecture uses depthwise separable convolutions, which factorize the standard convolution operation into a depthwise convolution and a pointwise convolution. This reduces the number of parameters in the network and makes it more efficient.

In Keras, MobileNetV2 is available as a pre-trained model, which can be used for transfer learning. By setting include_top=False, we can remove the final fully-connected layer of the model and add our own custom layers on top of the pre-trained layers to train the model for a specific task

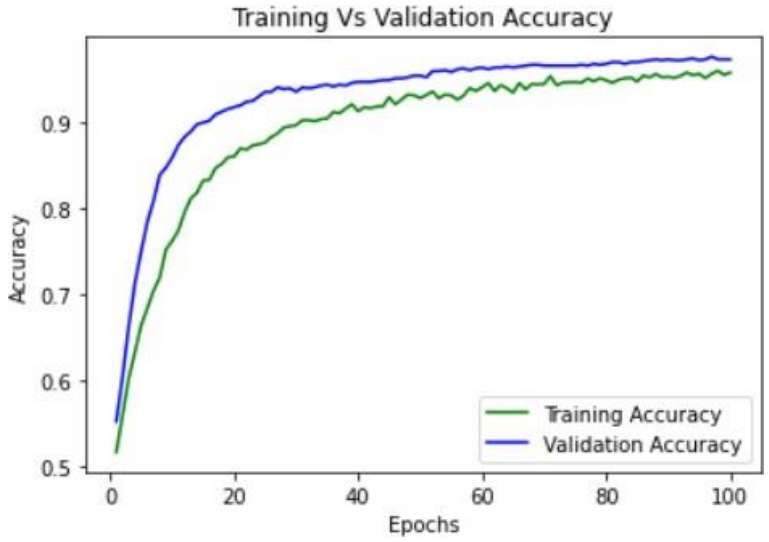

FIG 9 TRAINNG vs VALIDATION ACCURACY

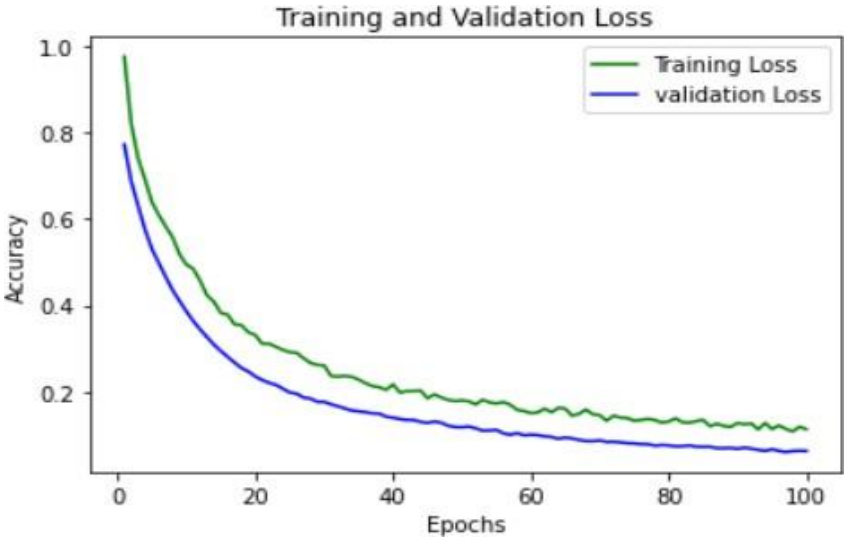

FIG 10 TRAINING vs VALIDATION LOSS

FACE DETECTION IMPLEMENTAION

Haar Cascade Classifier in Opencv:

The algorithm needs a lot of positive images (images of faces) and negative images(images without faces) to train the classifier. Then we need to extract features from it.

For this, haar features shown in below image are used. They are just like our

convolutional kernel. Each feature is a single value obtained by subtracting sum of

pixels under white rectangle from sum of pixels under black rectangle.

Now all possible sizes and locations of each kernel is used to calculate plenty of

features. (Just imagine how much computation it needs? Even a 24x24 window results over 160000 features). For each feature calculation, we need to find sum of pixels underwhite and black rectangles. To solve this, they introduced the integral images. Itsimplifies calculation of sum of pixels, how large may be the number of pixels, to anoperation involving just four pixels. Nice, isn't it? It makes things super-fast.

But among all these features we calculated, most of them are irrelevant. For example, consider the image below. Top row shows two good features. The first feature selected seems to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks. The second feature selected relies on the property that the eyes are darker than the bridge of the nose. But the same windows applying on cheeks or any other place is irrelevant. So how do we select the best features out of 160000+features? It is achieved by Adaboost.

For this, we apply each and every feature on all the training images. For each feature, it finds the best threshold which will classify the faces to positive and negative. But obviously, there will be errors or misclassifications. We select the features with minimum error rate, which means they are the features that best classifies the face and non-face images. (The process is not as simple as this. Each image is given an equalweight in the beginning. After each classification, weights

of misclassified images are increased. Then again same process is done. New error rates are calculated. Also new weights. The process is continued until required accuracy or error rate is achieved or required number of features are found).

Final classifier is a weighted sum of these weak classifiers. It is called weak because it alone can't classify the image, but together with others forms a strong classifier. The 38 paper says even 200 features provide detection with 95% accuracy. Their final setup had around 6000 features. (Imagine a reduction from 160000+ features to 6000 features. That is a big gain).

So now you take an image. Take each 24x24 window. Apply 6000 features to it. Check if it is face or not. Isn't it a little inefficient and time consuming? Yes, it is.

In an image, most of the image region is non-face region. So it is a better idea to have a simple method to check if a window is not a face region. If it is not, discard it in a single shot. Don't process it again. Instead focus on region where there can be a face. This way, we can find more time to check a possible face region.

Haar cascade works as a classifier. It classifies positive data points → that are part of our detected object and negative data points → that don't contain our object. Haar cascades are fast and can work well in real-time. Haar cascade is not as accurate as modern object detection techniques are.

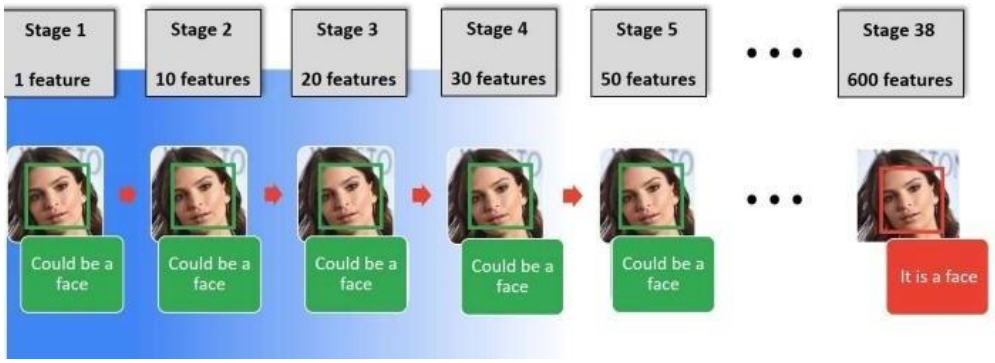

FIG 11 HAAR CASCADE CLASSIFIER

Evaluation of the Model

1. Data Set for testing the model

Testing dataset organized into one folder (Real-Fake-test) and contains 951 of Real-Fake dataset, (JPG) format, different from the images that used in original dataset for training; they are images of two classifications of real, fake-mid, fake-hard, fake-easy distributed as in the following table:

DISTRIBUTION OF IMAGES IN THE TEST DIRECTORY

CATEGORIES	NO.OF TESTING IMAGES	IMAGE SIZE
REAL	477	256 x 256 pixels
FAKE	474	256 x 256 pixels

TABLE 2 DISTRIBUTION OF IMAGES IN TEST DATASET

After training and evaluating the model on the validation, the network is tested using 951 Real-Fake dataset images, among them, 477 real, 474 fake.

Testing the models was done through loading the test images and predicting their classes using the model.predict_classes() function, probabilities of each image belonging to a specific class were calculated, by the following classification: real, fake. Note the classification rates of network during testing, it was a surprise, the probability of results in the classification for our model: MobileNet, was 97%, correct respectively for all test images.

TESTING THE MODEL

```
from keras.preprocessing import image
import numpy as np
def check_fakes(path,category_type):
  predictor = {}
  path= os.path.join(path,category_type)
  for img in os.listdir(path):
    try:
      img = image.load_img(os.path.join(path,img),target_size=(160,160))
      img = image.img_to_array(img)
      img = np.expand_dims(img,axis=0)
      img = img / 255.0
      prediction = model.predict(img)
      if prediction > 0.5:
        prediction_class = 1
      else:
        prediction_class = 0
      result = categories[prediction_class]
      if result not in predictor:
        predictor[result] = 1
      else:
        predictor[result] += 1
    except Exception as e:
      pass
  return predictor
```

This code is for testing a machine learning model that classifies images into two categories - "real" or "fake". It uses Keras, a popular deep learning framework, and numpy, a library for numerical computing in Python.

The function check_fakes takes two arguments - path and category_type. Path is the path to the directory containing the images to be tested, and category_type is either "real" or "fake", depending on the category of images being tested.

The first step in the function is to create an empty dictionary called predictor. This dictionary will be used to keep track of the number of images classified as "real" and "fake".

The function then uses os.path.join to create a path to the directory containing the images of the specified category type. It loops through all the images in the directory using os.listdir.

For each image, the code tries to load it using the Keras function image.load_img, and resizes it to 160x160 pixels using target_size=(160,160). The image is then converted to a numpy array using image.img_to_array.

Next, the array is expanded along the first axis using np.expand_dims, which adds an extra dimension to the array to represent the batch size. This is necessary because the Keras model expects input in the shape (batch_size, height, width, channels).

The array is then normalized by dividing it by 255.0, which scales the pixel values to the range 0-1.

The model is then used to predict the category of the image using model.predict. If the predicted probability of the image belonging to the "fake" category is greater than 0.5, the prediction_class variable is set to 1, indicating that the image is fake. Otherwise, prediction_class is set to 0, indicating that the image is real.

The result of the prediction is then mapped to the corresponding category using the categories dictionary. If the predicted category is not already in the predictor dictionary, it is added with a count of 1. Otherwise, the count for that category is incremented.

Finally, the predictor dictionary is returned, which contains the number of images classified as "real" and "fake" in the specified directory.

It is important to note that the success of this testing code depends on the accuracy of the underlying machine learning model. If the model is inaccurate, the results of the testing code will also be inaccurate. Additionally, the testing code assumes that all images in the specified directory are either "real" or "fake", and that the directory structure is organized in a specific way. If these assumptions are not met, the code may produce unexpected results.

TEST RESULTS

```
check_fakes(test_dir,categories[1])
```

```
{'real': 19, 'spoof': 455}
```

```
[ ] check_fakes(test_dir,categories[0])
```

```
{'real': 471, 'spoof': 6}
```

```
[ ] (19+6)/(19+455+471+6)*100
```

```
2.6288117770767614
```

```
[ ] 100-2.6288
```

```
97.3712
```

FIG 12 TEST RESULTS

CHAPTER 6

RESULTS, DISCUSSION AND COMPARISON

We trained our custom model on the training dataset using ,5171 images for a total of 100 epochs, the results were as follows:

Criterion	MobileNet
Training Accuracy	95%
Validation Accuracy	97.5%
Training loss	1.1%
Validation loss	0.6%
Testing Accuracy	97.3%

TABLE 3 MODEL METRICS

RESULTS

This study presents a deep learning approach for detecting if an image is real or faked. We accomplished this using the proposed pre-trained models: MobileNet architecture, these model were able to correctly predict Real-Fake images with a balanced accuracy approximately 97% on the validation and the test set after only training for 100 epochs, due to the techniques such as Adam optimization, data augmentation, dropout and others, it was possible to enhance the accuracy of model without sacrificing training efficiency, because one of the problems with machine learning, including deep learning, is overfitting. Overfitting occurs when the trained model does not generalize well to unseen cases but fits the training data well. This becomes more apparent when the training sample size is small. Assessment of the plot training can be used to assess the possibility of overfitting. From the curve, it is apparent that the data loss is similar for both validation and training datasets. If there were overfitting, the loss on the training data would be much greater than that of the validation data. In addition, for this reason, the cases were split two groups

(training, validation). It is conceivable that the use of larger training datasets, additional image augmentation methods, and additional machine learning approaches with more ensembles could improve this result.

COMPARISONS

Criterion	ResNet50	MobileNet	InceptionV3
Training Accuracy	94%	95%	93%
Validation Accuracy	95.3%	97.5%	96%
Training loss	1.7%	1.1%	1.3%
Validation loss	1.0%	0.6%	0.8%
Testing Accuracy	96.4%	97.3%	96.8%

TABLE 4 COMPARISONS

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENT

The advance of image capability and image generation techniques have now provided the ability to create security-less and convincing fake face images.

The challenging nature of data, whether in terms of visual perception or algorithm discovery, is present in recent works. The main problem that has not yet been taken into consideration is the assessment the overall capability of existing fake face detection techniques. in order to answer the question of generalizability, in this work, we have trained dataset using 5171 images for total of 100 epochs, and get the MOBILNET model is the best model of network architectures with 95% training accuracy, 97.8% validation accuracy, training loss 1.1, validation loss 0.6, and testing accuracy 97%. It was development by using Google Colab and Python language that supported for a deep learning machine.

In the authentication system, where the identification is done through face recognition, this customized model is capable to detect the spoofing attack during face recognition. This model works for photo print and mobile photo spoof attacks.

In further research, the model can be extended to detect mask and video attack during face recognition.

CHAPTER 8

REFERENCES

1. Al Barsh, Y. I., et al. (2020). "MPG Prediction Using Artificial Neural Network." International Journal of Academic Information Systems Research (IJAISR) 4(11): 7-16.

2. Alajrami, E., et al. (2019). "Blood Donation Prediction using Artificial Neural Network." International Journal of Academic Engineering Research (IJAER) 3(10): 1-7.

3. Alajrami, E., et al. (2020). "Handwritten Signature Verification using Deep Learning." International Journal of Academic Multidisciplinary Research (IJAMR) 3(12): 39-44.

4. Al-Araj, R. S. A., et al. (2020). "Classification of Animal Species Using Neural Network." International Journal of Academic Engineering Research (IJAER) 4(10): 23-31.

5. Al-Atrash, Y. E., et al. (2020). "Modeling Cognitive Development of the Balance Scale Task Using ANN." International Journal of Academic Information Systems Research (IJAISR) 4(9): 74-81.

6. Alghoul, A., et al. (2018). "Email Classification Using Artificial Neural Network." International Journal of Academic Engineering Research (IJAER) 2(11): 8-14.

7. Abu Nada, A. M., et al. (2020). "Age and Gender Prediction and Validation through Single User Images Using CNN." International Journal of Academic Engineering Research (IJAER) 4(8): 21-24.

8. Abu Nada, A. M., et al. (2020). "Arabic Text Summarization Using AraBERT Model Using Extractive Text Summarization Approach." International Journal of Academic Information Systems Research

(IJAISR) 4(8): 6-9.

9. Abu-Saqer, M. M., et al. (2020). "Type of Grapefruit Classification Using Deep Learning." International Journal of Academic Information Systems Research (IJAISR) 4(1): 1-5.

10. Afana, M., et al. (2018). "Artificial Neural Network for Forecasting Car Mileage per Gallon in the City." International Journal of Advanced Science and Technology 124: 51-59.

11. Al-Araj, R. S. A., et al. (2020). "Classification of Animal Species Using Neural Network." International Journal of Academic Engineering Research (IJAER) 4(10): 23-31.

12. Al-Atrash, Y. E., et al. (2020). "Modeling Cognitive Development of the Balance Scale Task Using ANN." International Journal of Academic Information Systems Research (IJAISR) 4(9): 74-81.

13. Alghoul, A., et al. (2018). "Email Classification Using Artificial Neural Network." International Journal of Academic Engineering Research (IJAER) 2(11): 8-14.

14. Al-Kahlout, M. M., et al. (2020). "Neural Network Approach to Predict Forest Fires using Meteorological Data." International Journal of Academic Engineering Research (IJAER) 4(9): 68-72.

15. Alkronz, E. S., et al. (2019). "Prediction of Whether Mushroom is Edible or Poisonous Using Back-propagation Neural Network." International Journal of Academic and Applied Research (IJAAR) 3(2): 1-8.

16. Al-Madhoun, O. S. E.-D., et al. (2020). "Low Birth Weight Prediction Using JNN." International Journal of Academic Health and Medical Research (IJAHMR) 4(11): 8-14.

17. Al-Massri, R., et al. (2018). "Classification Prediction of SBRCTs Cancers Using Artificial Neural Network." International Journal of Academic Engineering Research (IJAER) 2(11): 1-7.

18. Al-Mobayed, A. A., et al. (2020). "Artificial Neural Network for Predicting Car

Performance Using JNN." International Journal of Engineering and Information Systems (IJEAIS) 4(9): 139-145.

19. Al-Mubayyed, O. M., et al. (2019). "Predicting Overall Car Performance Using Artificial Neural Network." International Journal of Academic and Applied Research (IJAAR) 3(1): 1-5.

20. Alshawwa, I. A., et al. (2020). "Analyzing Types of Cherry Using Deep Learning." International Journal of Academic Engineering Research (IJAER) 4(1): 1-5.

21. Al-Shawwa, M., et al. (2018). "Predicting Temperature and Humidity in the Surrounding Environment Using Artificial Neural Network." International Journal of Academic Pedagogical Research (IJAPR) 2(9):

1-6.

22. Ashqar, B. A., et al. (2019). "Plant Seedlings Classification Using Deep Learning." International Journal of Academic Information Systems Research (IJAISR) 3(1): 7-14.

23. Bakr, M. A. H. A., et al. (2020). "Breast Cancer Prediction using JNN." International Journal of Academic Information Systems Research (IJAISR) 4(10): 1-8.

24. Barhoom, A. M., et al. (2019). "Predicting Titanic Survivors using Artificial Neural Network." International Journal of Academic Engineering Research (IJAER) 3(9): 8-12.