

SIES(NERUL) COLLEGE OF ARTS, SCIENCE AND COMMERCE

NAAC RE-ACCREDITED 'A' GRADE

SRI CHANDRASEKARENDRA SARASWATHY VIDHYAPURAM

PLOT 1-C SECTOR V

, NERUL NAVI MUMBAI -400706



A Project Report on

Face Anti-Spoofing

Submitted

By:

KARTHIK JAYAN NAIR

Roll NO: 09

Seat Number :

MSC CS PART 1

Under the Esteemed guidance

of

Dr.Rajeshri Shinkar

SIES(NERUL) COLLEGE OF ARTS, SCIENCE AND COMMERCE

NAAC RE-ACCREDITED 'A' GRADE

SRI CHANDRASEKARENDRA SARASWATHY VIDHYAPURAM

PLOT 1-C SECTOR V

, NERUL NAVI MUMBAI -400706



CERTIFICATE

This is to certify that the project entitled, **“Face Anti spoofing System”**, is bonafied work of **Karthik Jayan Nair** of Masters Of Science in computer Science (part 1) submitted in partial fulfillment of the requirements for the award of degree of **MASTER OF SCIENCE in COMPUTER SCIENCE** from University of Mumbai. It is also to clarify that this is the original work of the candidate done during the academic year 2022-23

Roll no: 09

Subject: Advanced Machine Learning

DR.Rajeshri Shinkar

Project guide

Date : _____

Table of Content

SRNO	TOPIC NAME	
1	Abstract	
2	Introduction and objective	
3	Detail Description of Techniques used	
4	Implementation	
5	Image of Application/Result	
6	Conclusion	
7	Limitation	
8	Future Scope	
9	Reference	

ABSTRACT

As there are many biometric modalities deployed for the security in the recent years. the face recognition and the voice recognition has great attention among the researchers. Iris, signature passwords and normal fingerprints are used for the security in many applications. these biometric identifiers have its own advantages and disadvantages. Critical issue addressed in the face recognition system is that they are attacker by different attacks like photos and videos. This paper details the different techniques available for detecting the spoofing in face recognition system. This paper also describes the database which is used by different researcher. The parameters used for evaluating the method is discussed at the end of the paper. This paper gives an idea to provide a comprehensive overview on the work that has been carried over the last decades in the emerging field of anti -spoofing. Keywords- Spoof attack, face recognition system, Biometric system, Attacks.

INTRODUCTION

Biometrics utilize physiological, such as fingerprint, face, and iris, or behavioural characteristics, such as typing rhythm and gait, to uniquely identify or authenticate an individual. As biometric systems are widely used in real-world applications including mobile phone authentication and access control, biometric spoof, or Presentation Attack (PA) are becoming a larger threat, where a spoofed biometric sample is presented to the biometric system and attempted to be authenticated. Since face is the most accessible biometric modality, there have been many different types of PAs for faces including print attack, replay attack, 3D masks, etc. As a result, conventional face recognition systems can be very vulnerable to such PAs. In order to develop a face recognition system that is invulnerable to various types of PAs, there is an increasing demand on designing a robust face anti-spoofing (or PA detection) system to classify a face sample as live or spoof before recognizing its identity. Previous approaches to tackle face anti-spoofing can be categorized in three groups. The first is the texture-based methods, which discover discriminative texture characteristics unique to various attack mediums. Due to a lack of an explicit correlation between pixel intensities and different types of attacks, extracting robust texture features is challenging. The second is the motionbased methods that aim at classifying face videos based on detecting movements of facial parts, e.g.,

eye blinking and lip movements. These methods are suitable for static attacks, but not dynamic attacks such as replay or mask attacks. The third is image quality and reflectance-based methods, which design features to capture the superimposed illumination and noise information to the spoof images

Objective

The primary objective of face anti-spoofing systems is to prevent fraudulent attempts to access secure systems or devices by spoofing or manipulating facial recognition systems. The use of facial recognition technology has become increasingly widespread in various fields, including security, banking, and mobile devices. However, these systems can be vulnerable to spoofing, where a fraudster presents a fake or manipulated face image to the system to gain unauthorized access.

The objective of face anti-spoofing systems is to detect and prevent such spoofing attempts by using various techniques, including image analysis and machine learning algorithms. The system can analyze the face image presented to it and determine if it is a genuine face or a spoofed image. The goal is to make the system more secure and accurate in identifying genuine users and preventing fraudulent access.

The use of face anti-spoofing systems can help organizations protect their sensitive data and prevent financial losses due to fraud. It can also help improve user experience by reducing the risk of false positives and making the authentication process more seamless and efficient. In summary, the primary objective of face anti-spoofing systems is to enhance the security and reliability of facial recognition systems and prevent fraudulent access to sensitive information.

DETAIL DESCRIPTION OF TECHNIQUE

Opencv module

OpenCV is a Python open-source library, which is used for computer vision in Artificial intelligence, Machine Learning, face recognition, etc.

In OpenCV, the CV is an abbreviation form of a computer vision, which is defined as a field of study that helps computers to understand the content of the digital images such as photographs and videos.

The purpose of computer vision is to understand the content of the images. It extracts the description from the pictures, which may be an object, a text description, and three-dimension model, and so on. For example, cars can be facilitated with computer vision, which will be able to identify and different objects around the road, such as traffic lights, pedestrians, traffic signs, and so on, and acts accordingly.

Computer vision allows the computer to perform the same kind of tasks as humans with the same efficiency. There are a two main task which are defined below:

- **Object Classification** - In the object classification, we train a model on a dataset of particular objects, and the model classifies new objects as belonging to one or more of your training categories.
- **Object Identification** - In the object identification, our model will identify a particular instance of an object - for example, parsing two faces in an image and tagging one as person 1 and person2

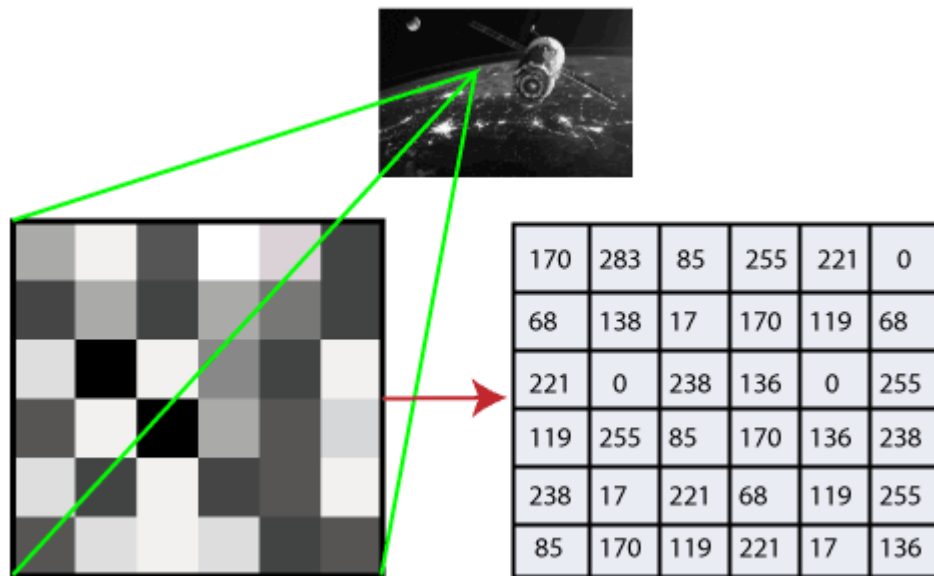
How OpenCV Works

In this tutorial, we will learn how computers perform image recognition.

How does computer recognize the image?

Human eyes provide lots of information based on what they see. Machines are facilitated with seeing everything, convert the vision into numbers and store in the memory. Here the question

arises how computer convert images into numbers. So the answer is that the pixel value is used to convert images into numbers. A pixel is the smallest unit of a digital image or graphics that can be displayed and represented on a digital display device.



The picture intensity at the particular location is represented by the numbers. In the above image, we have shown the pixel values for a grayscale image consist of only one value, the intensity of the black color at that location.

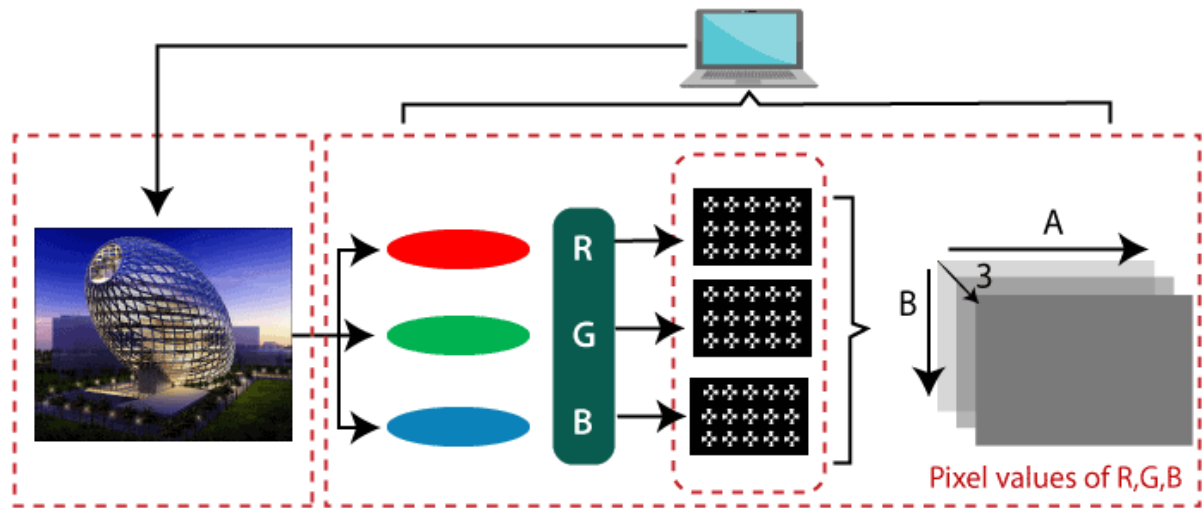
There are two common ways to identify the images:

1. Grayscale

Grayscale images are those images which contain only two colors black and white. The contrast measurement of intensity is black treated as the weakest intensity, and white as the strongest intensity. When we use the grayscale image, the computer assigns each pixel value based on its level of darkness.

2. RGB

An RGB is a combination of the red, green, blue color which together makes a new color. The computer retrieves that value from each pixel and puts the results in an array to be interpreted.



Face recognition using openCV

Basic Concept of HAAR Cascade Algorithm

The HAAR cascade is a machine learning approach where a cascade function is trained from a lot of positive and negative images. Positive images are those images that consist of faces, and negative images are without faces. In face detection, image features are treated as numerical information extracted from the pictures that can distinguish one image from another.

We apply every feature of the algorithm on all the training images. Every image is given equal weight at the starting. It finds the best threshold which will categorize the faces to positive and negative. There may be errors and misclassifications. We select the features with a minimum error rate, which means these are the features that best classifies the face and non-face images.

All possible sizes and locations of each kernel are used to calculate the plenty of features.

HAAR-Cascade Detection in OpenCV

OpenCV provides the trainer as well as the detector. We can train the classifier for any object like cars, planes, and buildings by using the OpenCV. There are two primary states of the cascade image classifier first one is training and the other is detection.

OpenCV provides two applications to train cascade classifier **opencv_haartraining** and **opencv_traincascade**. These two applications store the classifier in the different file format.

For training, we need a set of samples. There are two types of samples:

- **Negative sample:** It is related to non-object images.
- **Positive samples:** It is a related image with detect objects.

A set of negative samples must be prepared manually, whereas the collection of positive samples are created using the **opencv_createsamples** utility.

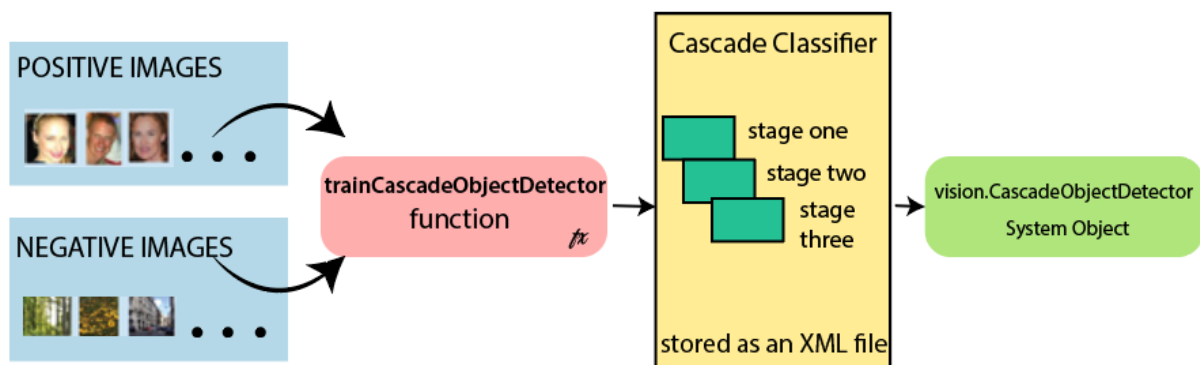
Negative Sample

Negative samples are taken from arbitrary images. Negative samples are added in a text file. Each line of the file contains an image filename (relative to the directory of the description file) of the negative sample. This file must be created manually. Defined images may be of different sizes.

Positive Sample

Positive samples are created by `opencv_createsamples` utility. These samples can be created from a single image with an object or from an earlier collection. It is important to remember that we require a large dataset of positive samples before you give it to the mentioned utility because it only applies the perspective transformation.

Cascade Classifier



Here we will discuss detection. OpenCV already contains various pre-trained classifiers for face, eyes, smile, etc. Those XML files are stored in **opencv/data/haarcascades/** folder. Let's understand the following steps:

Step - 1

First, we need to load the necessary XML classifiers and load input images (or video) in grayscale mode.

Step -2

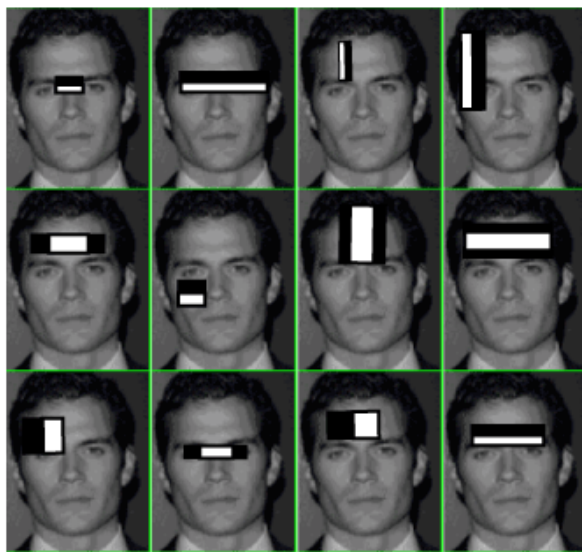
After converting the image into grayscale, we can do the image manipulation where the image can be resized, cropped, blurred, and sharpen if required. The next step is image segmentation; identify the multiple objects in the single image, so the classifier quickly detects the objects and faces in the picture.

Step - 3

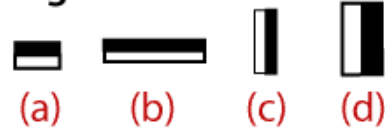
The haar-Like feature algorithm is used to find the location of the human faces in frame or image. All the Human faces have some common universal properties of faces like the eye region is darker than it's neighbor's pixels and nose region is more bright than the eye region.

Step -4

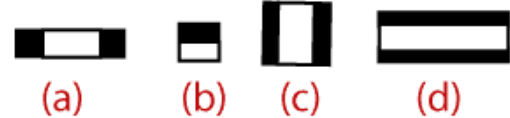
In this step, we extract the features from the image, with the help of edge detection, line detection, and center detection. Then provide the coordinate of x, y, w, h, which makes a rectangle box in the picture to show the location of the face. It can make a rectangle box in the desired area where it detects the face.



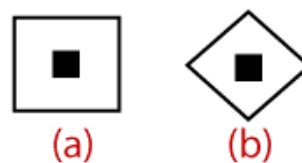
Edge Features



Line Features



Center-surround Features



Face recognition using OpenCV

Face recognition is a simple task for humans. Successful face recognition tends to effective recognition of the inner features (eyes, nose, mouth) or outer features (head, face, hairline). Here the question is that how the human brain encode it?

David Hubel and **Torsten Wiesel** show that our brain has specialized nerve cells responding to unique local feature of the scene, such as lines, edges angle, or movement. Our brain combines the different sources of information into the useful patterns; we don't see the visual as scatters. If we define face recognition in the simple word, "Automatic face recognition is all about to take out those meaningful features from an image and putting them into a useful representation then perform some classification on them".

The basic idea of face recognition is based on the geometric features of a face. It is the feasible and most intuitive approach for face recognition. The first automated face recognition system was described in the position of eyes, ears, nose. These positioning points are called features vector (distance between the points).

The face recognition is achieved by calculating the **Euclidean** distance between feature vectors of a probe and reference image. This method is effective in illumination change by its nature, but it has a considerable drawback. The correct registration of the marker is very hard.

The face recognition system can operate basically in two modes:

- **Authentication or Verification of a facial image-**

It compares the input facial image with the facial image related to the user, which is required authentication. It is a 1x1 comparison.

- **Identification or facial recognition**

It basically compares the input facial images from a dataset to find the user that matches that input face. It is a 1xN comparison.

There are various types of face recognition algorithms, for example:

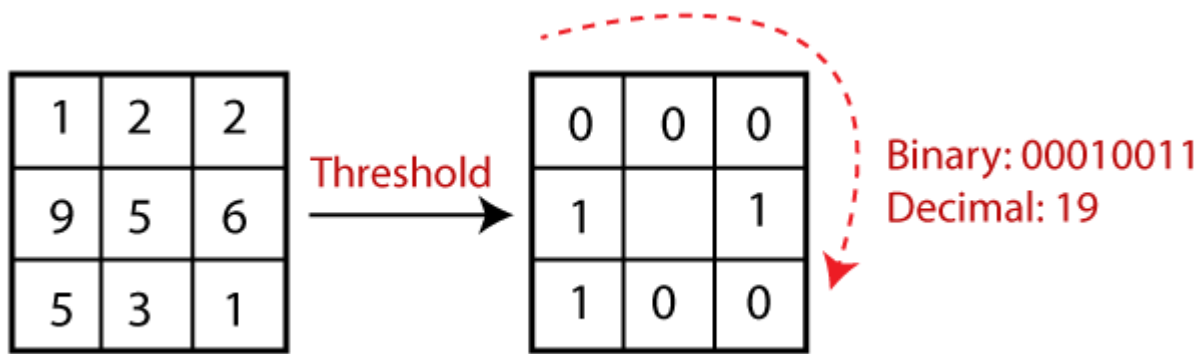
- **Eigenfaces (1991)**
- **Local Binary Patterns Histograms (LBPH) (1996)**
- **Fisherfaces (1997)**
- **Scale Invariant Feature Transform (SIFT) (1999)**
- **Speed Up Robust Features (SURF) (2006)**

Each algorithm follows the different approaches to extract the image information and perform the matching with the input image. Here we will discuss the Local Binary Patterns Histogram (LBPH) algorithm which is one of the oldest and popular algorithm.

Introduction of LBPH

Local Binary Pattern Histogram algorithm is a simple approach that labels the pixels of the image thresholding the neighborhood of each pixel. In other words, LBPH summarizes the local structure in an image by comparing each pixel with its neighbors and the result is converted into a binary number. It was first defined in 1994 (LBP) and since that time it has been found to be a powerful algorithm for texture classification.

This algorithm is generally focused on extracting local features from images. The basic idea is not to look at the whole image as a high-dimension vector; it only focuses on the local features of an object.



In the above image, take a pixel as center and threshold its neighbor against. If the intensity of the center pixel is greater-equal to its neighbor, then denote it with 1 and if not then denote it with 0.

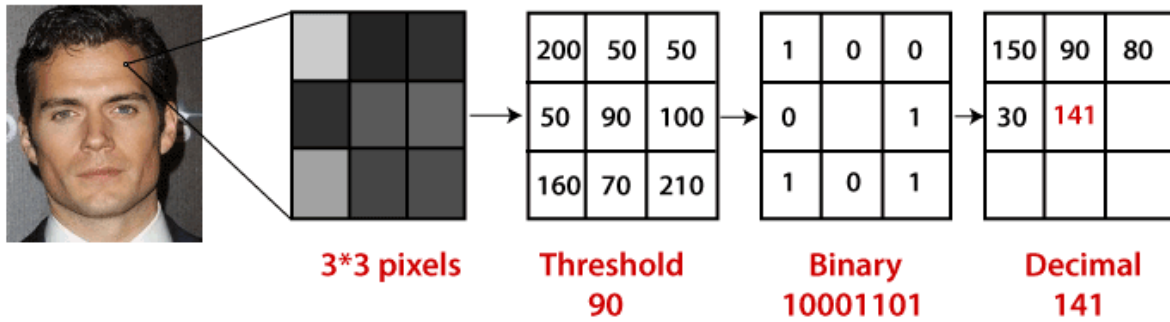
Let's understand the steps of the algorithm:

1. Selecting the Parameters: The LBPH accepts the four parameters:

- **Radius:** It represents the radius around the central pixel. It is usually set to 1. It is used to build the circular local binary pattern.
- **Neighbors:** The number of sample points to build the circular binary pattern.
- **Grid X:** The number of cells in the horizontal direction. The more cells and finer grid represents, the higher dimensionality of the resulting feature vector.
- **Grid Y:** The number of cells in the vertical direction. The more cells and finer grid represents, the higher dimensionality of the resulting feature vector.

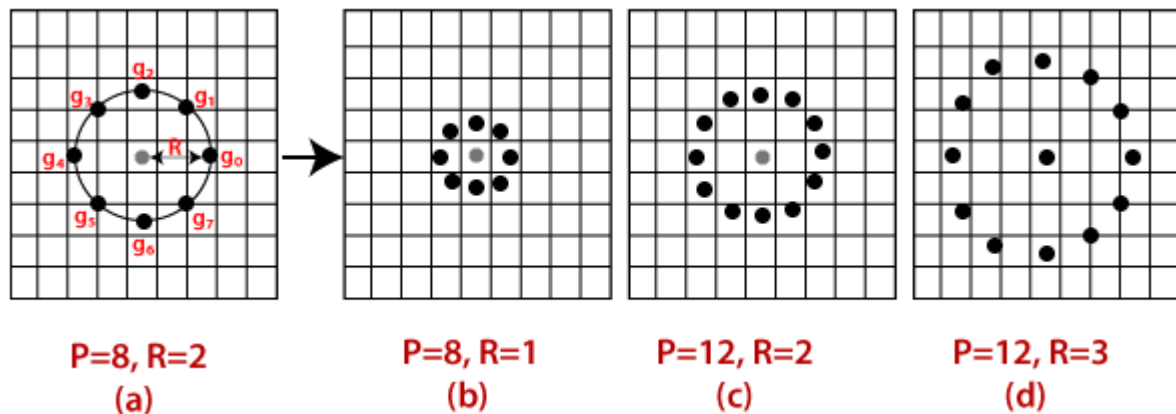
2. Training the Algorithm: The first step is to train the algorithm. It requires a dataset with the facial images of the person that we want to recognize. A unique ID (it may be a number or name of the person) should provide with each image. Then the algorithm uses this information to recognize an input image and give you the output. An Image of particular person must have the same ID. Let's understand the LBPH computational in the next step.

3. Using the LBP operation: In this step, LBP computation is used to create an intermediate image that describes the original image in a specific way through highlighting the facial characteristic. The parameters **radius** and **neighbors** are used in the concept of sliding window.

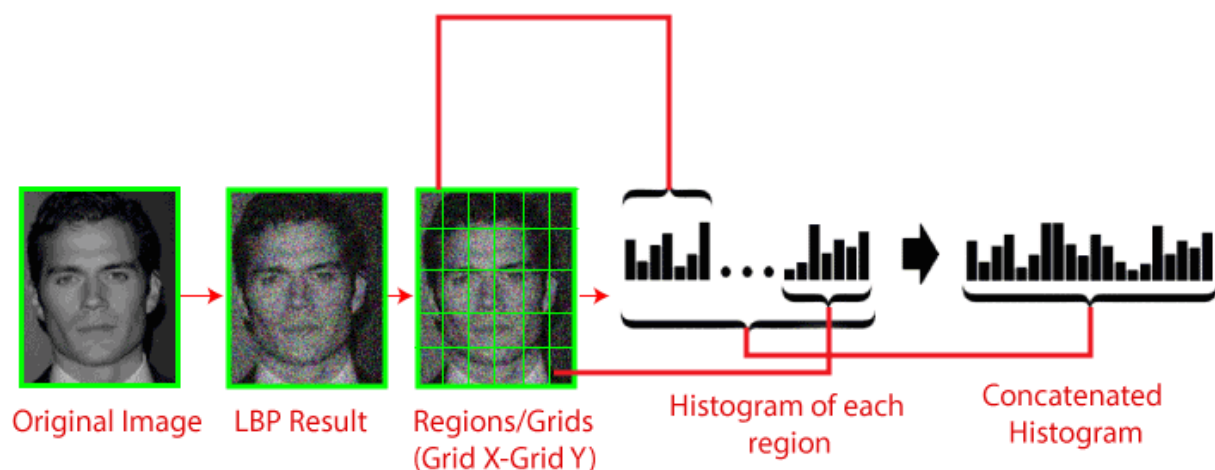


To understand in a more specific way, let's break it into several small steps:

- Suppose the input facial image is grayscale.
- We can get part of this image as a window of 3x3 pixels.
- We can use the 3x3 matrix containing the intensity of each pixel (0-255).
- Then, we need to take the central value of the matrix to be used as a threshold.
- This value will be used to define the new values from the 8 neighbors.
- For every neighbor of the central value (threshold), we set a new binary value. The value 1 is set for equal or higher than the threshold and 0 for values lower than the threshold.
- Now the matrix will consist of only binary values (skip the central value). We need to take care of each binary value from each position from the matrix line by line into new binary values (10001101). There are other approaches to concatenate the binary values (clockwise direction), but the final result will be the same.
- We convert this binary value to decimal value and set it to the central value of the matrix, which is a pixel from the original image.
- After completing the LBP procedure, we get the new image, which represents better characteristics of the original image.



4. Extracting the Histograms from the image: The image is generated in the last step, we can use the **Grid X** and **Grid Y** parameters to divide the image into multiple grids, let's consider the following image:



- We have an image in grayscale; each histogram (from each grid) will contain only 256 positions representing the occurrence of each pixel intensity.
- It is required to create a new bigger histogram by concatenating each histogram.

5. Performing face recognition: Now, the algorithm is well trained. The extracted histogram is used to represent each image from the training dataset. For the new image, we perform steps again and create a new histogram. To find the image that matches the given image, we just need to match two histograms and return the image with the closest histogram.

- There are various approaches to compare the histograms (calculate the distance between two histograms), for example: **Euclidean distance**, **chi-square**, **absolute value**, etc. We can use the Euclidean distance based on the following formula:

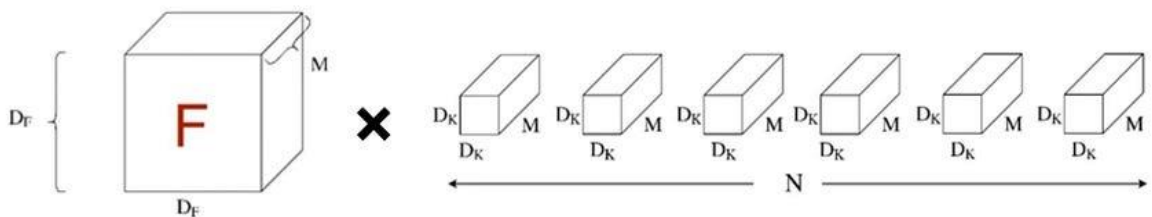
$$D = \sqrt{\sum_{i=1}^n (\text{hist } 1_i - \text{hist } 2_i)^2}$$

- The algorithm will return ID as an output from the image with the closest histogram. The algorithm should also return the calculated distance that can be called **confidence** measurement. If the confidence is lower than the threshold value, that means the algorithm has successfully recognized the face.

MobileNet for Object detection

1. Depth-wise separable convolution

- The Depth-wise separable convolution is comprising of two layers, the depth-wise convolution, and the point-wise convolution. Basically the first layer is used to filter the input channels and the second layer is used to combine them to create a new feature.
- **1.1 Depth-wise convolution**
- The depth-wise convolutions are used to apply a single filter into each input channel. This is different from a standard convolution in which the filters are applied to all of the input channels.
- Let's take a **standard convolution**,

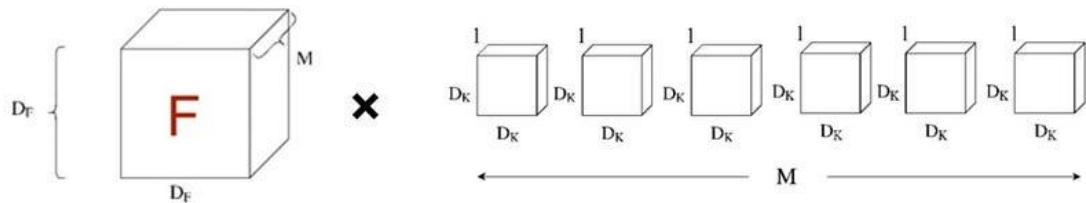


-
- Standard Convolution
- From the above image, the computational cost can be calculated as :

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F$$

-
- Standard Convolution Cost
- Where D_F is the special dimensions of the input feature map and D_K is the size of the convolution kernel. Here M and N are the number of input and output channels respectively.

- For a standard convolution, the computational cost depends multiplicatively on the number of input and output channels and on the spatial dimensions of the input feature map and convolution kernel.
- In case of depthwise convolution, as seen in the below image, contains an input feature map of dimension $D_F \times D_F$ and M number of kernels of channel size 1.



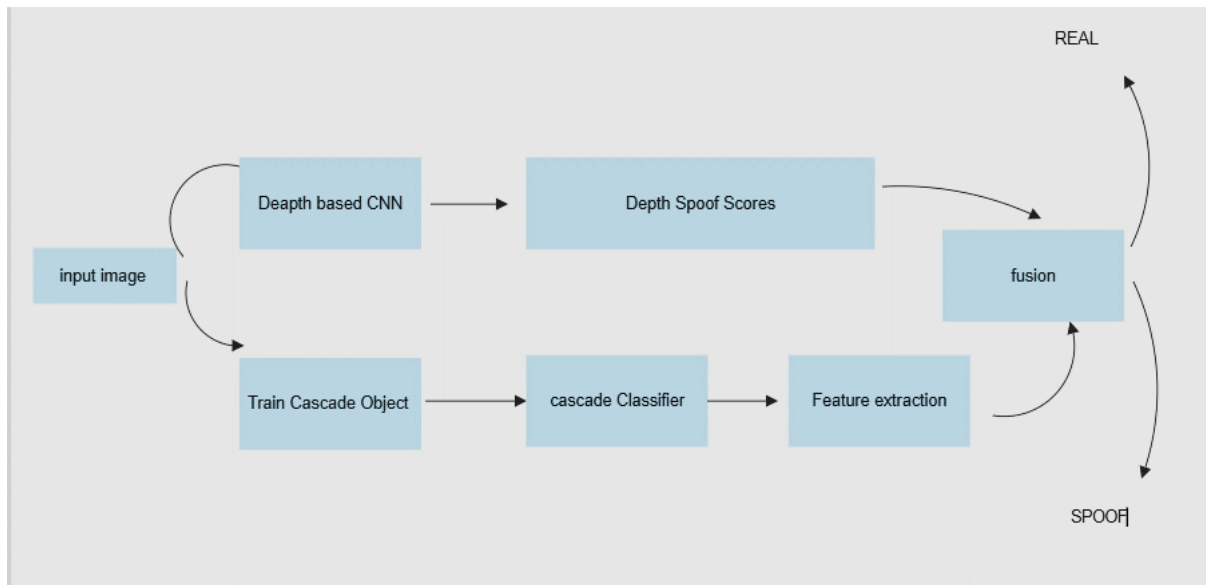
-
- Depth-wise Convolution
- As per the above image, we can clearly see that the total computational cost can be calculated as:

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F$$

-
- Depth-Wise Convolution cost
- However, this method is only used to filter the input channel.

IMPLEMENTATION

Flowchart



Code

<https://siescms.sharepoint.com/:u:/s/MSCCSPART12022-23/Ed6sHjfP75BCgoHCkCAfjywBQM3s-z3dYyyjATlvZ78mKA?e=HcR4GQ>

Data set :-

<https://siescms.sharepoint.com/:f:/s/MSCCSPART12022-23/EirfbSelCBRPm3KHc8qFk4oBE05GQTYeVy8CzHEJpRA0uA?e=ZiGKm0>

Python ui code

```
import cv2
from tensorflow.keras.preprocessing.image import img_to_array
import os
import numpy as np
from tensorflow.keras.models import model_from_json

root_dir = os.getcwd()
```

```

# Load Face Detection Model
face_cascade =
cv2.CascadeClassifier("antispoofing_models/haarcascade_frontalface_default.xml")
# Load Anti-Spoofing Model graph
json_file = open('antispoofing_models/antispoofing_model.json','r')
loaded_model_json = json_file.read()
json_file.close()
model = model_from_json(loaded_model_json)
# load antispoofing model weights
model.load_weights('antispoofing_models/antispoofing_model.h5')
print("Model loaded from disk")
# video.open("http://192.168.1.101:8080/video")
# vs = VideoStream(src=0).start()
# time.sleep(2.0)

video = cv2.VideoCapture(0)
while True:
    try:
        ret,frame = video.read()
        gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
        faces = face_cascade.detectMultiScale(gray,1.3,5)
        for (x,y,w,h) in faces:
            face = frame[y-5:y+h+5,x-5:x+w+5]
            resized_face = cv2.resize(face,(160,160))
            resized_face = resized_face.astype("float") / 255.0
            # resized_face = img_to_array(resized_face)
            resized_face = np.expand_dims(resized_face, axis=0)
            # pass the face ROI through the trained liveness detector
            # model to determine if the face is "real" or "fake"
            preds = model.predict(resized_face)[0]
            print(preds)
            if preds> 0.5:
                label = 'spooof'
                cv2.putText(frame, label, (x,y - 10),

```

```

        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 2)
cv2.rectangle(frame, (x, y), (x+w,y+h),
              (0, 0, 255), 2)
else:
    label = 'real'
    cv2.putText(frame, label, (x,y - 10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,255,0), 2)
    cv2.rectangle(frame, (x, y), (x+w,y+h),
                  (0, 255, 0), 2)
cv2.imshow('frame', frame)
key = cv2.waitKey(1)
if key == ord('q'):
    break
except Exception as e:
    pass
video.release()
cv2.destroyAllWindows()

```

model code

Getting datasets from google drive to workspace

```
In [ ]: from google.colab import drive
drive.mount('/content/gdrive')
```

Mounted at /content/gdrive


```
In [ ]: !cp -r "/content/gdrive/MyDrive/final_antispoofing.zip" "/content"
```

```
In [ ]: import zipfile
archive = zipfile.ZipFile('/content/final_antispoofing.zip')
archive.extractall('/content')
```

Original VS New Directory Structure

```
In [ ]: from IPython import display
print("Original Vs New Dataset Structure")
display.Image('original_vs_new_dataset.png')
```

Original Vs New Dataset Structure

Out[4]: 

```
In [ ]: dataset_dir = '/content/final_antispoofing'
train_dataset_dir = '/content/final_antispoofing/train'
test_dataset_dir = '/content/final_antispoofing/test'
```

```
In [ ]: 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')
```

```
In [ ]: train_dir = '/content/antispoofing_dataset/train'
test_dir = '/content/antispoofing_dataset/test'
```

Step 2

```
In [ ]: import shutil
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

```
In [ ]: 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, category, img))
                    else:
                        shutil.copy(os.path.join(path_to_subject, img), os.path.join(test_dir, category, img))
```

```
In [ ]: train_test_splits(data_directory=dataset_dir)
```

Step 3

Dataset Exploration

```
In [ ]: categories = ['real','spooof']
```

```
In [ ]: 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 spooof directory
There are 4220 total images in training directory
-----Exploring Testing Datasets-----
There are 477 images in real directory
There are 474 images in spooof directory
There are 951 total images in testing directory
```

Step 4 : Dataset Visualization

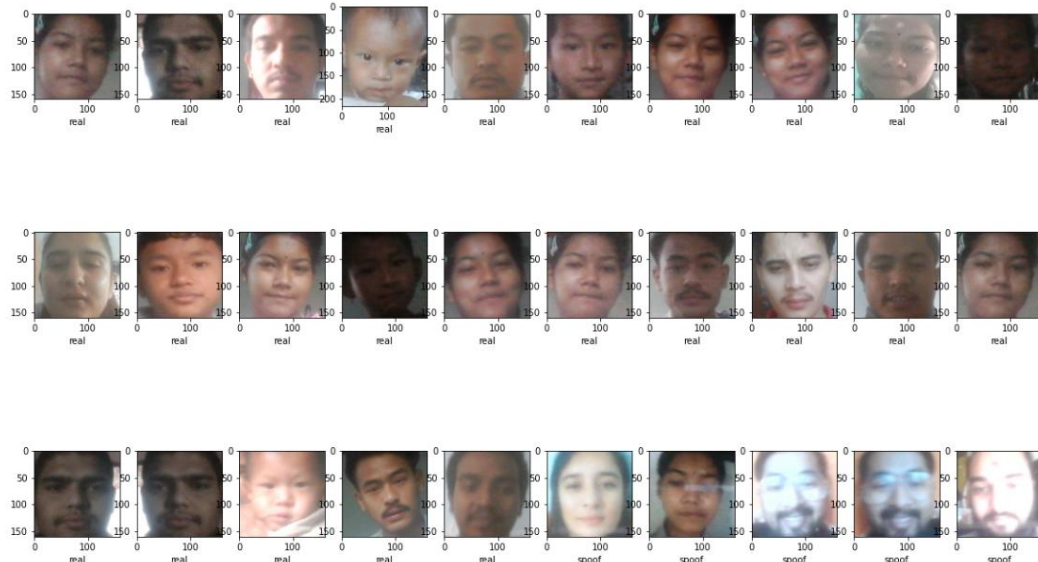
```
In [ ]: def get_images(data_dir,number_of_samples):
        image_path = []
        for category in categories:
            path = os.path.join(data_dir,category)
            i = 1
            for img in os.listdir(path):
                if i > number_of_samples:
                    break
      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

```
In [ ]: 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()
```

```
In [ ]: training_image_path = get_images(data_dir= train_dir,number_of_samples=25)
        print(training_image_path)
        print(len(training_image_path))
```

Training Dataset Visualization

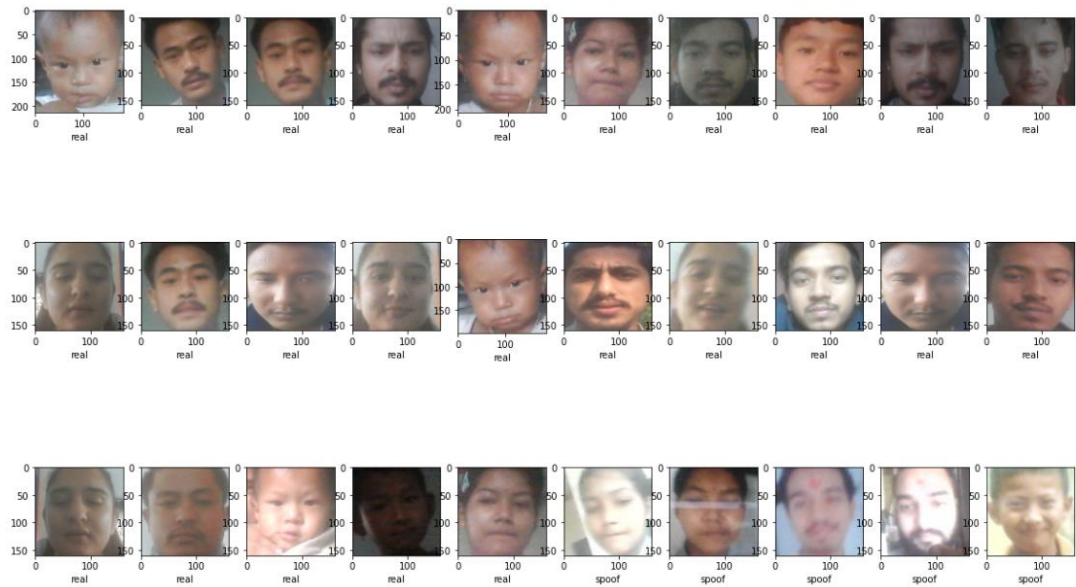
```
In [ ]: visualize_dataset(image_path=training_image_path,rows=5,cols=10)
```



Testing Dataset Visualization

```
In [ ]: testing_image_path = get_images(data_dir= test_dir,number_of_samples=25)
        print(testing_image_path)
        print(len(testing_image_path))
```

```
In [ ]: visualize_dataset(image_path=testing_image_path,rows=5,cols=10)
```



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

Step 1 Keras Framework

```
In [ ]: 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
```

Step 2

```
In [ ]: train_datagen = ImageDataGenerator(brightness_range=(0.8,1.2),rotation_range=30,width_shift_range=0.2,height_shift_range=0.2,fill_mode='nearest')
valid_datagen = ImageDataGenerator(rescale=1./255)
```

```
In [ ]: train_generator = train_datagen.flow_from_directory(train_dir,target_size=(160,160),color_mode='rgb',
class_mode='binary',batch_size=25,shuffle=True)
```

Found 4220 images belonging to 2 classes.

```
In [ ]: valid_generator = valid_datagen.flow_from_directory(test_dir,target_size=(160,160),color_mode='rgb',
class_mode='binary',batch_size=25)
```

Found 951 images belonging to 2 classes.

Step 3 Model Selection and Transfer Learning

```
In [ ]: mobilenet = MobileNetV2(weights="imagenet",include_top=False,input_tensor=Input(shape=(160,160,3)))

WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [96, 128, 160, 192, 224]. Weights for input shape (224, 224) will be loaded as the default.

In [ ]: mobilenet.trainable = False

In [ ]: output = Flatten()(mobilenet.output)
output = Dropout(0.3)(output)
output = Dense(units = 8,activation='relu')(output)
prediction = Dense(1,activation='sigmoid')(output)

In [ ]: model = Model(inputs = mobilenet.input,outputs = prediction)
model.summary()
```

Step 4 Compiling the models

```
In [ ]: # 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']
)
```

Step 5 Setting our model checkpoints

```
In [ ]: import os
os.mkdir('/content/model_weights/')

In [ ]: model_checkpoint = ModelCheckpoint('./model_weights/finalyearproject_antispoofing_model_{epoch:02d}-{val_accuracy:.6f}.h5', monitor

In [ ]: history = model.fit_generator(
    train_generator,
    steps_per_epoch = train_generator.samples // 25,
    validation_data = valid_generator,
    validation_steps = valid_generator.samples // 25,
    epochs = 100,
    callbacks=[model_checkpoint])
```

```
In [ ]: # serialize model to JSON
model_json = model.to_json()
with open("antispoofing_model_mobilenet.json", "w") as json_file:
    json_file.write(model_json)

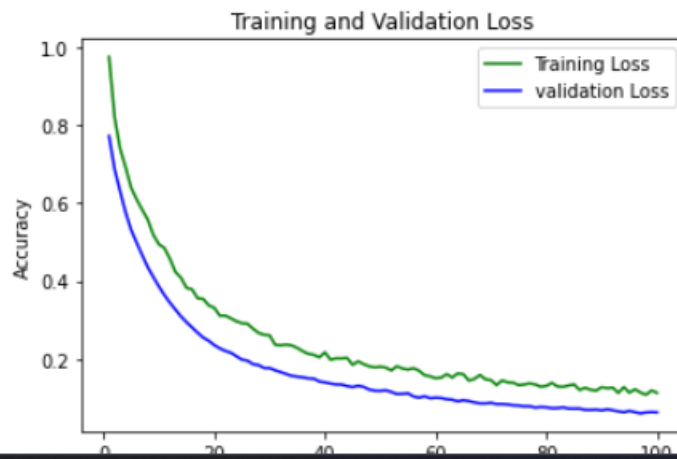
/usr/local/lib/python3.7/dist-packages/keras/utils/generic_utils.py:497: CustomMaskWarning: Custom mask layers and must override get_config. When loading, the custom mask layer must be passed to the custom_objects argument category=CustomMaskWarning)
```

```
In [ ]: import matplotlib.pyplot as plt
train_accuracy = history.history['accuracy']
validation_accuracy = history.history['val_accuracy']
epochs = range(1,101)
plt.plot(epochs,train_accuracy,'g',label='Training Accuracy')
plt.plot(epochs,validation_accuracy,'b',label='Validation Accuracy')
plt.title('Training Vs Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

Training Vs Validation Accuracy

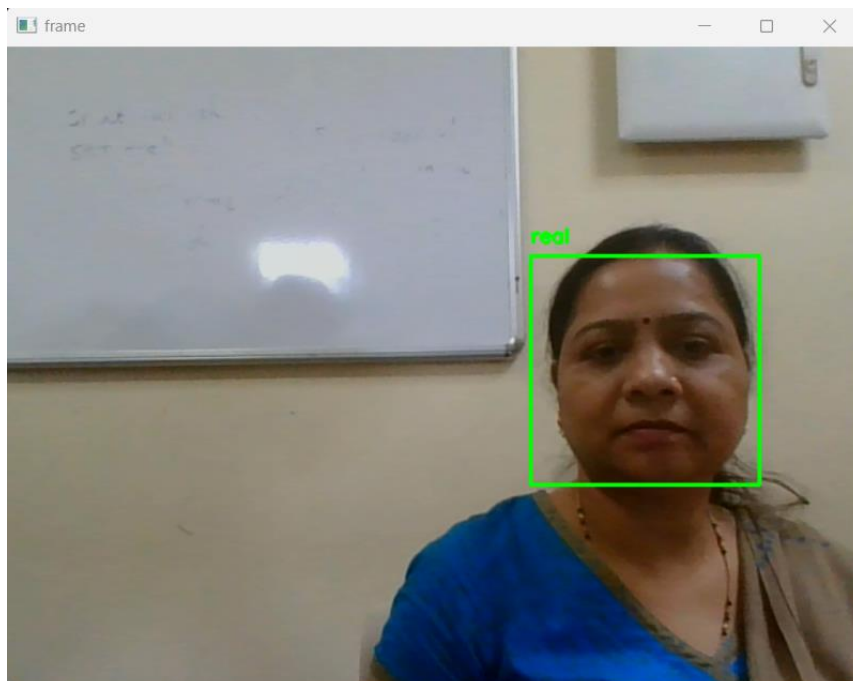


```
In [ ]: import matplotlib.pyplot as plt
train_loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1,101)
plt.plot(epochs,train_loss,'g', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

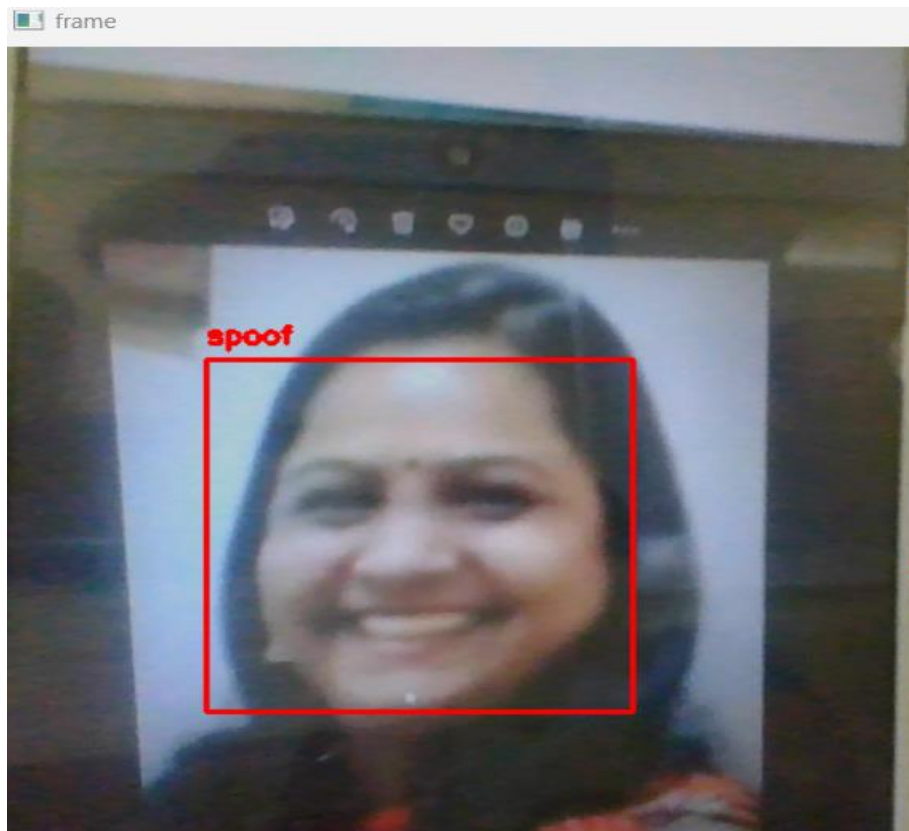


Result

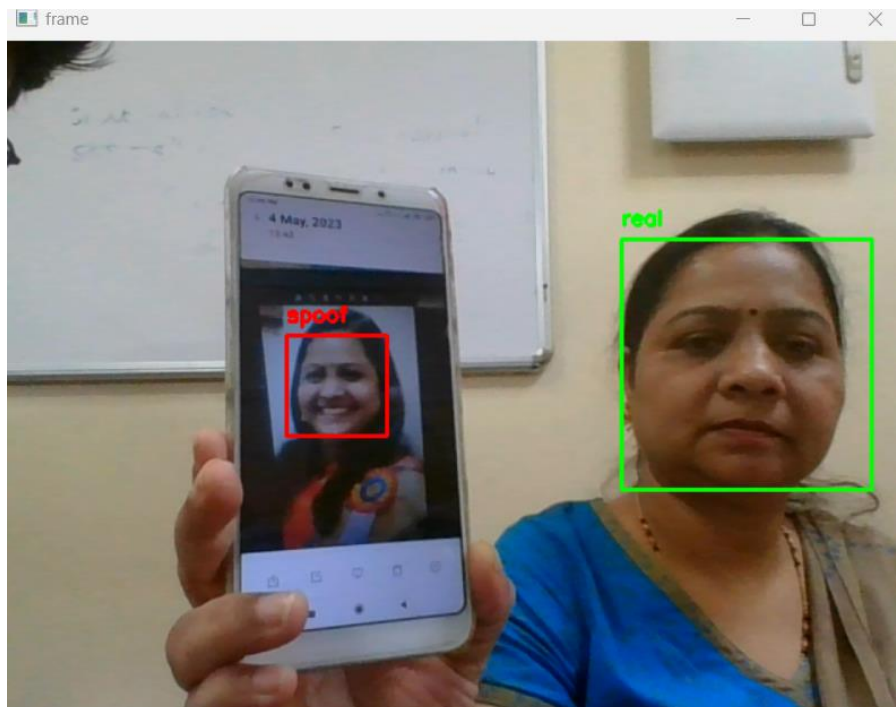
Output1:-



Output2:-

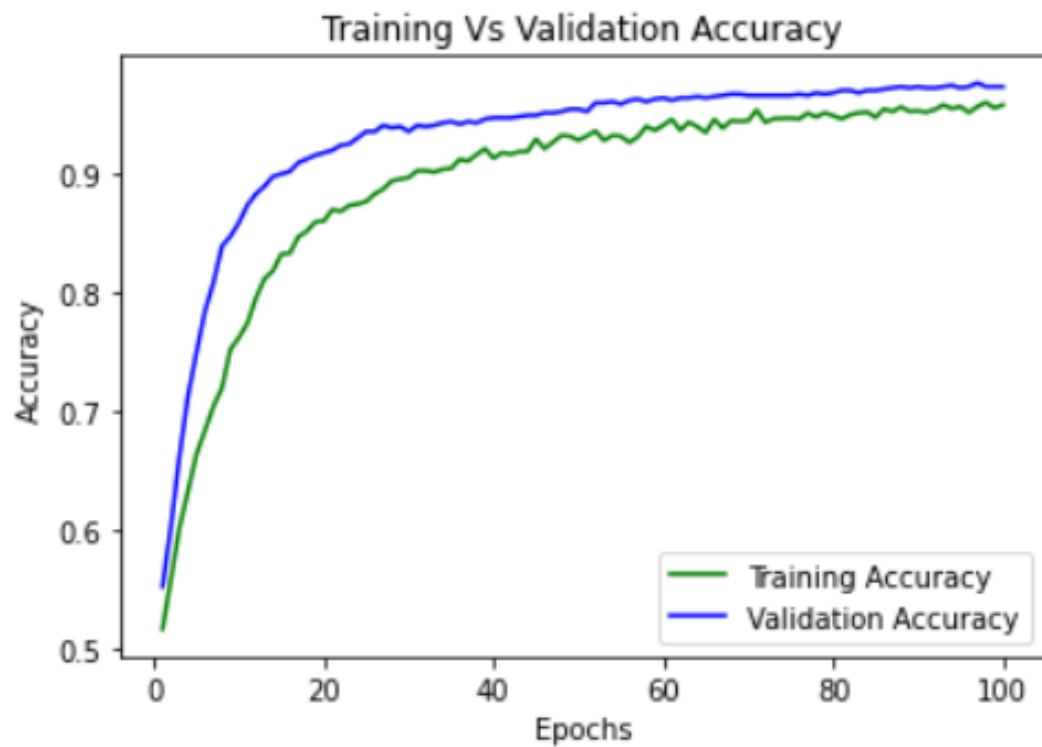


Output3:-

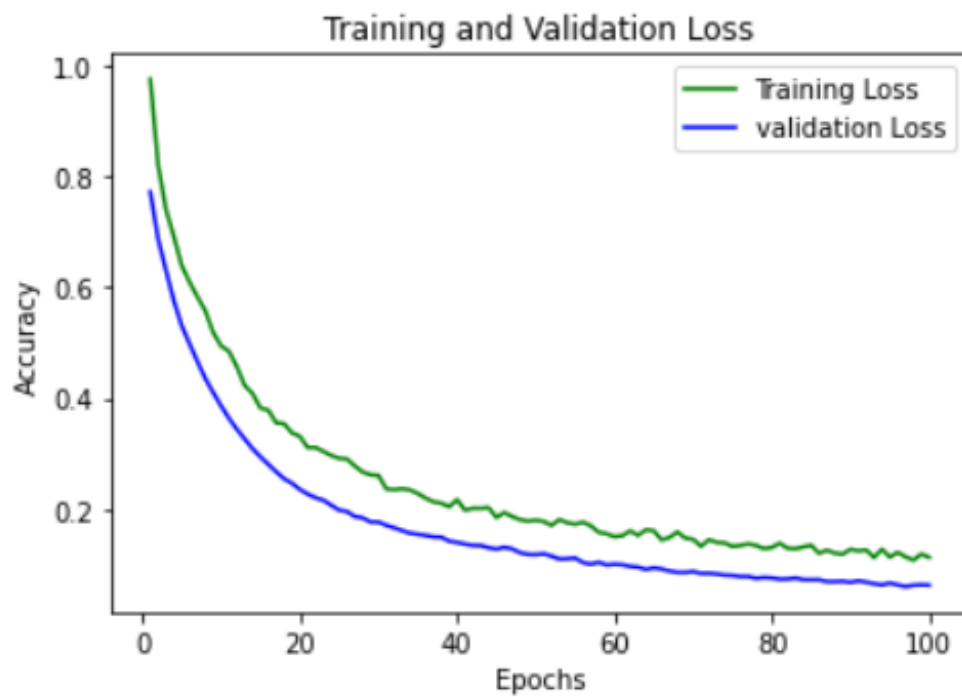


Conclusion:-

Graph1(Accuracy):-



Graph2:-



Accuracy Score:-

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

```
Out[44]: {'real': 19, 'spoof': 455}
```

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

```
Out[45]: {'real': 471, 'spoof': 6}
```

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

```
Out[47]: 2.6288117770767614
```

```
In [ ]: 100-2.6288
```

```
Out[48]: 97.3712
```

As You can see that the model gives 97.37 percent accuracy the accuracy is calculated by dividing true positive and true negative by all the value thus it gives the accuracy of the value and the to calculate the percentile reduce it by hundred.

As you must have understood by now, how each of the spoof preventive methods targets a particular attack and fail in the case of other attacks.

The Liveliness detection method only saves from the image attacks and fails for any other attacks.

The Depth detection method saves from the image as well as video attacks but fails for 3d print attacks.

Depth detection is still to be preferred if only a single method is to be deployed, as it prevents most attacks. Also, the attacks it prevents are the most common and can be performed easily.

But an ideal Anti-Spoofing system should not rely on a single method but must incorporate multiple methods working together to prevent spoofing attempts in as many cases as possible.

Limitations

The proposed anti-spoofing system prevents the popular attacks from bypassing authentication that is attempted using an image and even stops the video attacks used to bypass liveness detection systems. However, there are still situations where it fails, such as a colored 3d print of the face geometry can be used to bypass it.

And there is the 3d mask attack can be challenging to detect for any system because of how similar it makes the attacker's face to the authenticated person's face

Future scope

Face anti-spoofing systems are designed to detect and prevent fraudulent attempts to access secure systems or devices by spoofing or manipulating facial recognition systems. These systems have become increasingly important as the use of facial recognition technology continues to grow in various fields, including security, banking, and mobile devices.

The future scope of face anti-spoofing systems is bright, as the technology continues to evolve and improve. Here are some potential areas of development and application:

1. **Advanced algorithms:** Researchers are continually developing advanced algorithms for face anti-spoofing systems, including deep learning and artificial intelligence techniques. These algorithms will help improve the accuracy and reliability of the systems, making them more effective at detecting spoofing attempts.
2. **Multi-modal biometrics:** Face anti-spoofing systems can be combined with other biometric technologies, such as voice or iris recognition, to create multi-modal biometric systems. This can increase the security of the system and reduce the risk of false positives.
3. **Mobile devices:** As facial recognition becomes more prevalent in mobile devices, face anti-spoofing systems will become increasingly important to prevent unauthorized access to sensitive data.

4. Emerging technologies: Face anti-spoofing systems may be applied to emerging technologies, such as augmented reality and virtual reality, to prevent fraudulent attempts to manipulate facial recognition systems in these environments.
5. Industry-specific applications: Different industries have different requirements for face anti-spoofing systems. For example, the banking industry may require more stringent security measures than the retail industry. As such, there is scope for the development of industry-specific face anti-spoofing systems.

In summary, the future of face anti-spoofing systems is promising, with potential for continued advancements and applications in various fields.

References

Mobile net:

<https://www.youtube.com/watch?v=5JAZiue-fzY>
https://www.youtube.com/watch?v=tVgUap_BPZw
<https://www.youtube.com/watch?v=5JAZiue-fzY>

<https://medium.com/@godeep48/an-overview-on-mobilenet-an-efficient-mobile-vision-cnn-f301141db94d>

opencv

[OpenCV: Cascade Classifier](#)
[OpenCV Haar Cascades - PyImageSearch](#)
[Python | Haar Cascades for Object Detection - GeeksforGeeks](#)
<http://cvlab.cse.msu.edu/project-face-anti.html>
<http://cvlab.cse.msu.edu/pdfs/FaceAntiSpoofingUsingPatchandDepthBasedCNNs.pdf>
<https://acadpubl.eu/jsi/2017-117-20-22/articles/22/38.pdf>