**B.M.S College of Engineering**
**P.O. Box No.: 1908 Bull Temple Road,**
**Bangalore-560 019**

**DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING**

**Multi-disciplinary Project Report**
**on**

# OFFLINE BIOMETRIC VERIFICATION AND AUTHENTICATION

*Submitted by*

**Niranjan Hegde (1BM19IS103)**
**Samartha S (1BM19IS219)**

*Under the guidance of*

**Prof. Nalina V**
**Assistant Professor, BMSCE**

**BACHELOR OF ENGINEERING**
***in***
**INFORMATION SCIENCE AND ENGINEERING**

**April-2022 to July-2022**

**B. M. S. College of Engineering,**

**Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)

## Department of Information Science and Engineering

# CERTIFICATE

This is to certify that the project work entitled **“Offline Biometrics Verification and Authentication”** carried out by **Niranjan Hegde (1BM19IS103)** and **Samartha S (1BM19IS219)** who are bonafide students of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Information Science and Engineering** of the VisvesvarayaTechnological University, Belgaum during the year 2021. The project report has been approved as it satisfies the academic requirements in respect of **Multi-disciplinary Project (20IS6PWMPR)** work prescribed for the said degree.

Signature of the Guide

Prof. Nalina V
Assistant Professor
BMSCE, Bengaluru

Signature of the HOD

Dr. P Jayarekha
Professor & Head,
BMSCE, Bengaluru

## External Examiners

Name of the Examiner | Signature with date

1.________________________________ ___________________________

2. ________________________________ ____________________________

# B. M. S. COLLEGE OF ENGINEERING
## DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

## *DECLARATION*

We, **Niranjan Hegde (1BM19IS103)** and **Samartha S (1BM19IS219)**, students of 6th Semester, B.E, Department of Information Science and Engineering, B. M. S. College of Engineering, Bangalore, hereby declare that, this Project Work-1entitled **"Offline Biometrics Verification and Authentication"** has been carried out by us under the guidance of **Prof. Nalina V**, Assistant Professor, Department of ISE, B.M.S. College of Engineering, Bangalore during the academic semester April-2022 to July-2022

We also declare that to the best of our knowledge and belief, the development reported here is not part of any other report by any other students.

| **Name of Student** | **Signature** |
|---|---|
| Niranjan Hegde (1BM19IS103) | |
| Samartha S (1BM19IS219) | |

# TABLE OF CONTENTS

# 1. Introduction

## 1.1 Motivation

Biometric security technologies are now found in almost every aspect of modern life. Biometric data and/or authentication devices are now found in a wide range of sectors such as government, companies, libraries, universities, banks etc., and also in a wide variety of things, including passports, driver's license, laptops etc. In biometrics and document forensics, signature and fingerprint verification are two of the most difficult tasks. It involves identifying minute but critical details between real and fake signatures/fingerprints. These verification systems are mostly based on manual verification, in which a person examines and compares the given signature/prints to the test signature/prints, necessitating the development of a more advanced system that is computer based.

Deep Learning is quickly gaining traction as a leading field that has been successfully applied to a variety of fields, especially image processing. In our project, we would like to propose an offline signature and fingerprint verification and authentication system using a convolutional Siamese network. Siamese networks are twin networks with shared weights that can be trained to learn a feature space in which comparable observations are clustered together. With this project, we aim to achieve better accuracy than the majority of existing verification systems.

## 1.2 Scope of the Project

Biometric security technologies have been used for security related purposes since ancient times. Biometric data and/or authentication devices are being used in a wide range of sectors such as government, companies, libraries, universities, banks etc. They are also used to provide security to things like mobile phones, laptops, driving licenses, passports etc. Biometric systems are designed to recognise a person based on physiological or behavioral characteristics. The recognition in the first scenario is based on measurements of biological features such as the fingerprint, face, iris, and so on. The latter case is concerned with behavioral characteristics such as voice and signature. Verification and identification are the two most important contexts in which biometric systems are used. In the first scenario, a person asserts his or her identification and submits a biometric sample. The verification system's job is to make sure that the person is who he or she claims to be. In the identification scenario, a user submits a biometric sample, and the goal is to identify it among all the other users in the system.

There are various types of biometric technologies such as fingerprint, signature, iris detection, face recognition, voice recognition etc. Among these, signature and fingerprint are the most common and they have been used since ancient times. The fact that people are familiar with the use of signatures and fingerprints in their day to day lives also contributes to why these biometric technologies are popular. Signature and fingerprint verification is an essential and crucial task, and numerous efforts have been made to minimize the uncertainty associated with manual authentication process, making signature and fingerprint verification a significant research topic in the fields of AI, machine learning, deep learning and pattern recognition.

This project can be extended to e-KYC to authenticate users and to authenticate students writing online exams.

## 1.3 Problem statement

Verifying and authentication of users based on their signature and fingerprint.

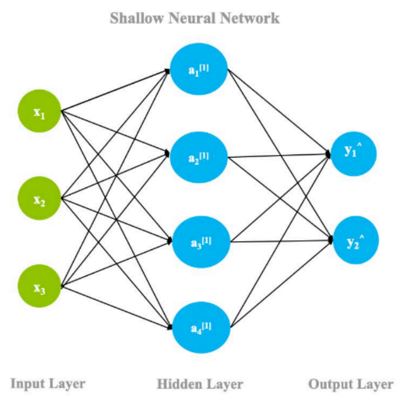
# 2. Literature Survey

We know the importance of biometric security systems and how they play a crucial role in our day-to-day lives. In our project, we aim at developing a model that can accurately distinguish between genuine and forged inputs. We plan on developing a convolutional siamese network to achieve our objective. The field of biometric verification is not new. It has been studied for many many years by many different researchers, and a lot of advancements have been made in the past few decades. In this section, we will take a look at some of the emerging technologies that have helped in the advancement of this field.

## Deep Learning

Deep learning is a type of machine learning that is trained on massive amounts of data and uses a large number of compute units to make predictions. The goal is to build a system that is similar to how humans think and learn, and hence the underlying architecture for Deep Learning was inspired by the structure of a human brain. Deep learning is entirely based on artificial neural networks. Similar to how the fundamental building blocks of a brain is the neuron, Deep learning architecture includes a computational unit called a perceptron that permits modeling of nonlinear functions. The perceptron receives a list of input signals and changes them into output signals in the same way that a neuron in the human brain transmits electrical pulses throughout our nervous system. The perceptron attempts to comprehend data representation by stacking multiple layers, each of which is responsible for understanding a different aspect of the input. Each layer of perceptrons is in charge of deciphering a distinct pattern in the data. The architecture is called artificial neural networks because a network of these perceptrons mimics the way neurons in the brain create a network. A neural network has three layers in its most basic form: input layer, hidden layer, and output layer. A shallow neural network is one that has only one hidden layer, as depicted in diagram 1.

Fig 1



## Convolutional Neural Network

The Convolutional Neural Network works by taking an image, assigning it a weightage depending on the image's different objects, and then distinguishing them from one another. In comparison to other deep learning algorithms, CNN requires very minimal data pre-processing. One of CNN's strongest features is that it uses primitive methods to train its classifiers, allowing it to learn the characteristics of the target object. CNNs are most commonly utilized in the field of pattern recognition within images. This enables us to encode image-specific properties into the architecture, making the network better suited for image-focused tasks while also lowering the number of parameters needed to set up the model.

## Siamese Neural Network

Neural networks perform well at almost every activity in the present deep learning era, but they rely on additional data to perform properly. However, we can't always rely on more data for specific problems like face recognition and signature verification; to handle these challenges, we have a new form of neural network architecture called Siamese Networks. Siamese Networks have gained popularity over the past few years due to their ability to learn from relatively little data. A Siamese Neural Network is a type of neural network architecture that has two or more subnetworks that are identical i.e. they have a similar configuration with the same parameters and weights, and the parameter updating is mirrored across both the sub-networks.

## One shot learning

One shot learning is a type of classification that correctly makes predictions when given only a single example of each new class.

To develop a model for one-shot image classification, we should construct a neural network that can discriminate between the class-identity of image pairings, which is the typical verification task for image recognition. The verification model learns to classify input pairs based on their likelihood of belonging to the same class. This model can then be used to evaluate new images against the test image, one for each unique class. For the one-shot task, the pairing with the greatest score according to the verification network is given the highest probability.

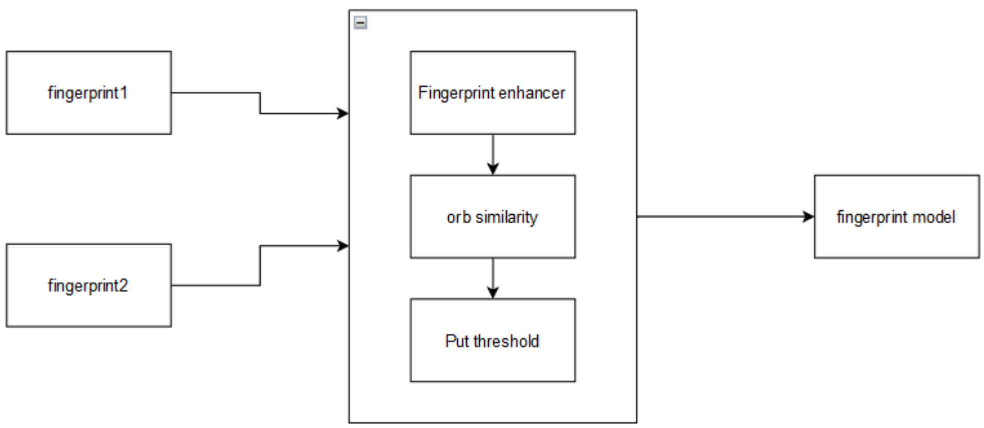
## Basic steps involved in verifying the biometric information

A. **Data Capture -** Capturing the images of the signature/fingerprint
B. **Pre-Processing** - Removal of any noise, normalization of the images, and getting all the data ready to be trained
C. **Feature extraction** - Extracting relevant features from the digital representation of the images
D. **Experimentation** - The proposed model is subjected to experimentation
E. **Performance evaluation** - The outcome of the proposed method is evaluated in terms of parameters like false rejection rate (FRR) of genuine images and the false acceptance rate (FAR) of forged images

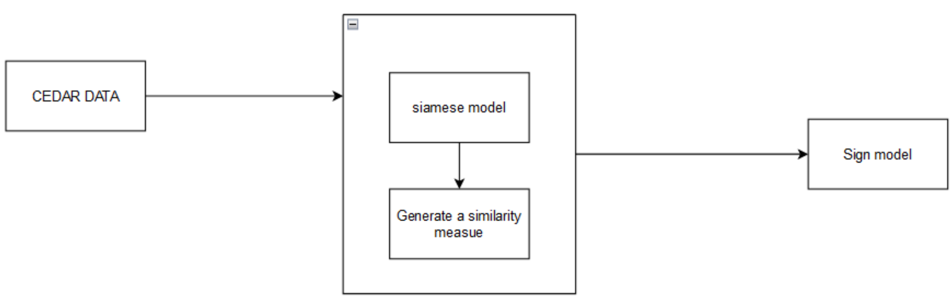
# 3. Design

## 3.1 High Level Design

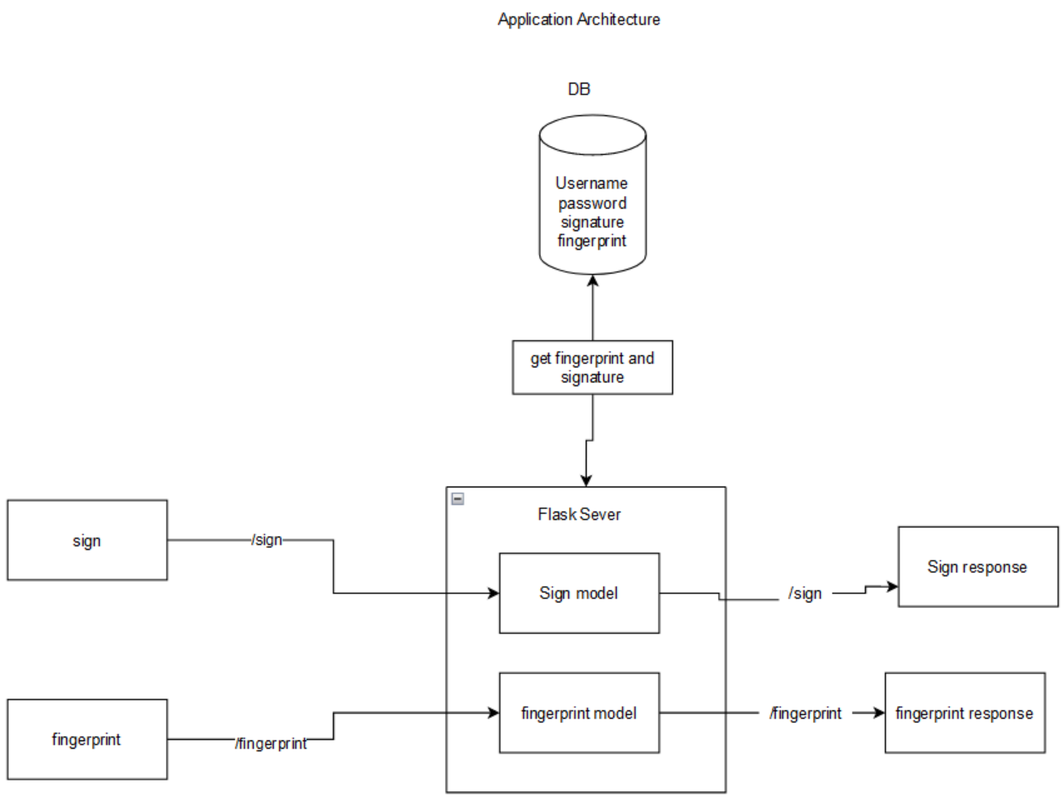
The following diagram shows the high level design for building the fingerprint verification model. We have 2 input fingerprint images. The fingerprint enhancer will enhance the images and then we pass the images to the orb similarity algorithm which will output the similarity score.



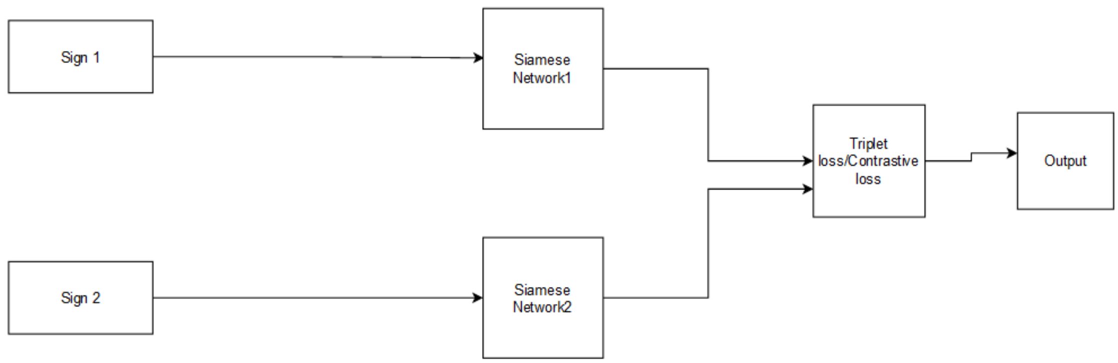

The following diagram shows the high level design for building the signature verification model. We feed the signature data to a siamese model and obtain a similarity score.



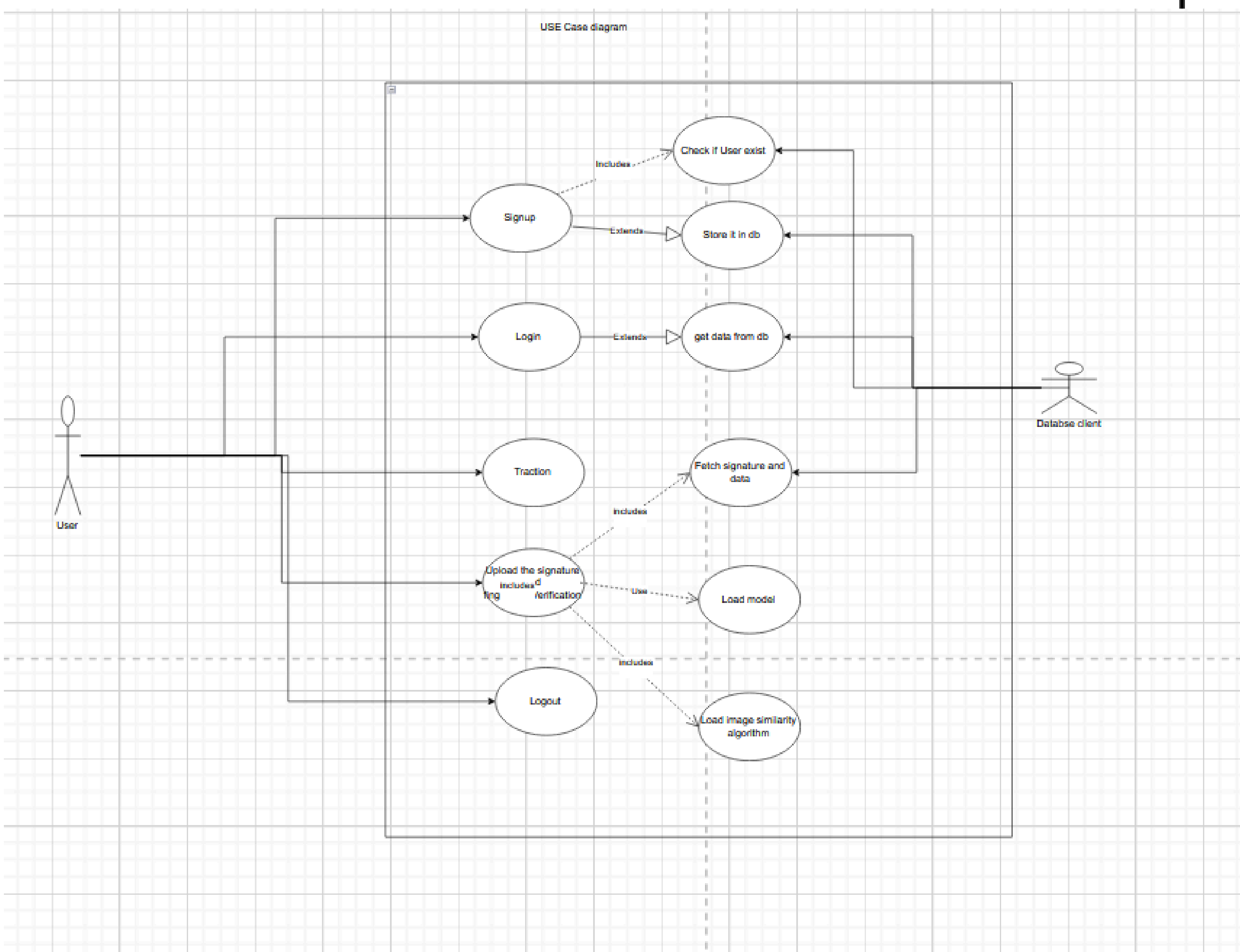
## 3.2 Detailed Design



Siamese Model architecture



## 3.3 Use Case Diagram



# 4. Implementation

## 4.1 Proposed methodology

In our project, we would like to propose an offline signature and fingerprint verification system using a convolutional Siamese network. Siamese networks are twin networks with shared weights that can be trained to learn a feature space in which comparable observations are clustered together.

We have used various concepts such as Deep Learning, Convolutional Neural Networks, Siamese Networks, ORB similarity, MongoDB, Flask and Web Development.

## 4.2 Algorithms used for implementation

- **ORB similarity:**

  ORB is a fusion of FAST keypoint detector and BRIEF descriptor with some added features to improve the performance. FAST is Features from Accelerated Segment Test used to detect features from the provided image. It also uses a pyramid to produce multiscale-features. Now it doesn’t compute the orientation and descriptors for the features, so this is where BRIEF comes in the role.

  ORB uses BRIEF descriptors but as the BRIEF performs poorly with rotation. So what ORB does is to rotate the BRIEF according to the orientation of keypoints. Using the orientation of the patch, its rotation matrix is found and rotates the BRIEF to get the rotated version. ORB is an efficient alternative to SIFT or SURF algorithms used for feature extraction, in computation cost, matching performance, and mainly the patents. SIFT and SURF are patented and you are supposed to pay them for its use. But ORB is not patented.

- **Siamese Network:**

  A Siamese Network consists of twin networks which accept distinct inputs but are joined by an energy function at the top. This function computes a metric between the highest level feature representation on each side. The parameters between the twin networks are tied. Weight tying guarantees that two extremely similar images are not mapped by each network to very different locations in feature space because each network computes the same function. The network is symmetric, so that whenever we present two distinct images to the twin networks, the top conjoining layer will compute the same metric as if we were to we present the same two images but to the opposite twins.

Intuitively instead of trying to classify inputs, a siamese network learns to differentiate between inputs, learning their similarity. The loss function used is usually a form of contrastive loss.

## 4.3 Tools and technologies used

- **Kaggle**: A data science platform which has collection of open source dataset and provides notebook service to build and train the model
- **Tensorflow**: A open source computational module in python for training and developing ML and DL models
- **Flask**: Flask is a small and lightweight Python web framework that provides useful tools and features that make creating web applications in Python easier
- **MongoDb**: Nosql database for storing user information
- **GCP**: Explored google cloud platform integrated with kaggle notebook for training the model
- **VS code**: To write and run python scripts
- **Git/GitHub**: For version control and collaboration with team

## 4.4 Code

**1.fingerprint similarity.py**

```
import cv2
```

```
import os

import sys

import numpy

import matplotlib.pyplot as plt

# from enhance import ImageEnhance

import fingerprint_enhancer

from skimage.morphology import skeletonize, thin


# os.chdir("/app/")


def removedot(invertThin):

    temp0 = numpy.array(invertThin[:])

    temp0 = numpy.array(temp0)

    temp1 = temp0/255

    temp2 = numpy.array(temp1)

    temp3 = numpy.array(temp2)


    enhanced_img = numpy.array(temp0)

    filter0 = numpy.zeros((10,10))

    W,H = temp0.shape[:2]

    filtersize = 6


    for i in range(W - filtersize):
```

```
        for j in range(H - filtersize):
            filter0 = temp1[i:i + filtersize,j:j + filtersize]


            flag = 0
            if sum(filter0[:,0]) == 0:
                flag +=1
            if sum(filter0[:,filtersize - 1]) == 0:
                flag +=1
            if sum(filter0[0,:]) == 0:
                flag +=1
            if sum(filter0[filtersize - 1,:]) == 0:
                flag +=1
            if flag > 3:
                            temp2[i:i  +  filtersize,  j:j  +  filtersize]  =
numpy.zeros((filtersize, filtersize))


    return temp2




def get_descriptors(img):
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    img = clahe.apply(img)
    img = fingerprint_enhancer.enhance_Fingerprint(img)
    img = numpy.array(img, dtype=numpy.uint8)
```

```
    # Threshold
    ret, img = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)
    # Normalize to 0 and 1 range
    img[img == 255] = 1


    #Thinning
    skeleton = skeletonize(img)
    skeleton = numpy.array(skeleton, dtype=numpy.uint8)
    skeleton = removedot(skeleton)
    # Harris corners
    harris_corners = cv2.cornerHarris(img, 3, 3, 0.04)
    harris_normalized = cv2.normalize(harris_corners, 0, 255, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_32FC1)
    threshold_harris = 125
    # Extract keypoints
    keypoints = []
    for x in range(0, harris_normalized.shape[0]):
      for y in range(0, harris_normalized.shape[1]):
        if harris_normalized[x][y] > threshold_harris:
          keypoints.append(cv2.KeyPoint(y, x, 1))
    # Define descriptor
    orb = cv2.ORB_create()
    # Compute descriptors
```

```
    _, des = orb.compute(img, keypoints)

    return (keypoints, des)



def similarity_orb(img1,img2):

    img1 = cv2.imread(img1, cv2.IMREAD_GRAYSCALE)

    kp1, des1 = get_descriptors(img1)


    img2 = cv2.imread(img2, cv2.IMREAD_GRAYSCALE)

    kp2, des2 = get_descriptors(img2)


    # Matching between descriptors

    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

             matches   =   sorted(bf.match(des1,   des2),   key=   lambda
match:match.distance)

    # Plot keypoints

    img4 = cv2.drawKeypoints(img1, kp1, outImage=None)

    img5 = cv2.drawKeypoints(img2, kp2, outImage=None)

    f, axarr = plt.subplots(1,2)

    axarr[0].imshow(img4)

    axarr[1].imshow(img5)

    plt.show()

    # Plot matches

       img3  =  cv2.drawMatches(img1,  kp1,  img2,  kp2,  matches,  flags=2,
```

```
outImg=None)

    plt.imshow(img3)

    plt.show()


    # Calculate score

    score = 0

    for match in matches:

      score += match.distance

    score_threshold = 33

    if score/len(matches) < score_threshold:

      print("Fingerprint matches.")

      return 1

    else:

      print("Fingerprint does not match.")

      return 0




# if __name__ == "__main__":

#   try:

#     main('./Datasets/101_1.tif','./Datasets/101_2.tif')

#   except:

#     raise
```

**2.signature_model.py**

```python
# -*- coding: utf-8 -*-

"""cedar-siamese-net(1).ipynb


Automatically generated by Colaboratory.


Original file is located at

https://colab.research.google.com/drive/1fnmgs5hf0BuVXgS68MI64VWPMJ5Q_TmY

"""


# This Python 3 environment comes with many helpful analytics libraries installed

# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python

# For example, here's several helpful packages to load


import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

```
# Input data files are available in the read-only "../input/" directory

# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory


import os

for dirname, _, filenames in os.walk('/kaggle/input'):

    for filename in filenames:

        print(os.path.join(dirname, filename))


# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"

# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session


"""import numpy as np

from keras import layers

from keras.layers import Input, Add, Multiply, Subtract, Dense, Activation, ZeroPadding2D, BatchNormalization, Flatten, Conv2D, AveragePooling2D, Dropout, MaxPooling2D, GlobalMaxPooling2D

from keras.layers import Dense, Dropout, Input, Lambda, Flatten, Convolution2D, MaxPooling2D, ZeroPadding2D

from keras.regularizers import l2

from keras.models import Model, load_model, Sequential

from keras.preprocessing import image
```

```
from keras.utils import layer_utils

from keras.utils.data_utils import get_file

from keras.applications.imagenet_utils import preprocess_input

"""



import numpy as np

from keras import layers

from keras.layers import Input, Add, Multiply, Subtract, Dense, Activation, ZeroPadding2D, BatchNormalization, Flatten, Conv2D, AveragePooling2D, Dropout, MaxPooling2D, GlobalMaxPooling2D

from keras.layers import Dense, Dropout, Input, Lambda, Flatten, Convolution2D, MaxPooling2D, ZeroPadding2D

from keras.regularizers import l2

from keras.models import Model, load_model, Sequential

from keras.preprocessing import image

from keras.utils import layer_utils

from keras.utils.data_utils import get_file

from keras.applications.imagenet_utils import preprocess_input



# Commented out IPython magic to ensure Python compatibility.

from IPython.display import SVG

from keras.utils.vis_utils import model_to_dot

# from keras.utils import plot_model

#from resnets_utils import *
```

```
from keras.initializers import glorot_uniform, he_normal

import scipy.misc

from matplotlib.pyplot import imshow

import matplotlib.pyplot as plt

# %matplotlib inline

import keras.backend as K

K.set_image_data_format('channels_last')

# K.set_learning_phase(1)

import scipy

import cv2

from keras.layers import subtract

import keras

from keras.layers import Lambda

import tensorflow as tf

from tensorflow.keras.optimizers import Adam,RMSprop


from tensorflow import image

from tensorflow.keras.utils import *


from PIL import Image,ImageOps


SIZE=(220,155)
```

```
def preprocess(img_input):

    img_input=cv2.resize(img_input,SIZE,interpolation=cv2.INTER_LINEAR )

#     img_input=cv2.bitwise_not(img_input)

    img_input=img_input/245

    return img_input


import os

img_path='../input/cedardataset/signatures/full_org/original_11_17.png'




img = load_img(img_path,target_size=(155,220))

    x = img_to_array(img)

    x=preprocess(x)


x.shape


from matplotlib.pyplot import imshow


imshow(x)


def euclidian(vects):

    X,y=vects
```

```
    return K.sqrt(K.sum(K.square(X-y),axis=1,keepdims=True))


def euclidean_distance_output_shape(shapes):

    shape1, shape2 = shapes

    return (shape1[0], 1)


def contrastive_loss(y, preds, margin=1):

    # explicitly cast the true class label data type to the predicted

    # class label data type (otherwise we run the risk of having two

    # separate data types, causing TensorFlow to error out)

    y = tf.cast(y, preds.dtype)

    # calculate the contrastive loss between the true labels and

    # the predicted labels

    squaredPreds = K.square(preds)

    squaredMargin = K.square(K.maximum(margin - preds, 0))

    loss = K.mean(y * squaredPreds + (1 - y) * squaredMargin)

    # return the computed contrastive loss to the calling function

    return loss


input_shape=(155,230,3)


def siamese(input_shape):

    model=Sequential()
```

```
                    model.add(Conv2D(96,     (11,11),     activation='relu',
name='conv11',input_shape=input_shape))

    model.add(BatchNormalization(epsilon=1e-06,  axis=1, momentum=0.9))

    model.add(MaxPooling2D((3,3), strides=(2, 2),padding='same'))



    model.add(Conv2D(256, (5,5), activation='relu', name='conv12'))

    model.add(BatchNormalization(epsilon=1e-06,  axis=1, momentum=0.9))

    model.add(MaxPooling2D((3,3), strides=(2, 2),padding='same'))

    model.add(Dropout(0.3))


                     model.add(Conv2D(384,     (3,3),     activation='relu',
name='conv13',input_shape=input_shape,padding='same'))

    model.add(Conv2D(256, (3,3), activation='relu', name='conv14'))

    model.add(MaxPooling2D((3,3), strides=(2, 2),padding='same'))

    model.add(Flatten())

                    model.add(Dense(1024,     kernel_regularizer=l2(0.0005),
activation='relu', kernel_initializer=glorot_uniform(seed=0)))

    model.add(Dropout(0.3))

    model.add(Dense(128, kernel_regularizer=l2(0.0005), activation='relu',
kernel_initializer=glorot_uniform(seed=0)))

    return model



#  model.add(Convolution2D(256,  5,  5,  activation='relu',  name='conv11',
```

```
subsample=(4, 4),
#                       init='he_normal'))
#      model.add(BatchNormalization(epsilon=1e-06,      mode=0,      axis=1,
momentum=0.9))
# model.add(MaxPooling2D((3,3), strides=(2, 2)))
# model.add(Dropout(0.3))
# model.add(ZeroPadding2D((2, 2)))


apply_sigmodel = siamese((155,220,3))


X_input1 = Input(shape=(155,220,3))
X_input2 = Input(shape=(155,220,3))


X_vect1 = apply_sigmodel(X_input1)
X_vect2 = apply_sigmodel(X_input2)


distance                          =                          Lambda(euclidian,
output_shape=euclidean_distance_output_shape)([X_vect1,X_vect2])


model = Model(inputs=[X_input1,X_input2], outputs=distance)


rms = RMSprop(learning_rate=1e-4, rho=0.9, epsilon=1e-08)


model.compile(loss=contrastive_loss,optimizer=rms,metrics=['accuracy'])
```

```
forg_list_1=[]

for i in range(20):

    p=[]

    for j in range(24):

                                                          img_path        =
'../input/cedardataset/signatures/full_forg/forgeries_'+str(i+1)      +'_'
+str(j+1)+'.png'

        img = load_img(img_path,target_size=(155,220))

        x = img_to_array(img)

        x=preprocess(x)

        p.append(x)

    forg_list_1.append(p)


org_list_1=[]

for i in range(20):

    p=[]

    for j in range(24):

                                                          img_path        =
'../input/cedardataset/signatures/full_org/original_'+str(i+1)        +'_'
+str(j+1)+'.png'

        img = load_img(img_path,target_size=(155,220))

        x = img_to_array(img)

        x=preprocess(x)

        p.append(x)
```

```
    org_list_1.append(p)


X_train1=[]

X_train2=[]

for i in range(10):

    count=24

    while(count>1):

        for j in range(0,count-1):

            X_train1.append(org_list_1[i][24-count])

            X_train2.append(org_list_1[i][24-count+j+1])

        count=count-1


for i in range(10):

    count=24

    while(count>1):

        for j in range(0,count-1):

            X_train1.append(org_list_1[i][24-count])

            X_train2.append(forg_list_1[i][24-count+j+1])

        count=count-1


len(X_train1)


X_train1=np.asarray(X_train1)
```

```
X_train2=np.asarray(X_train2)


12420


Y_train=np.array([])

for m in range(2760):

    Y_train=np.insert(Y_train,0,1)

for n in range(2760):

    Y_train=np.insert(Y_train,0,0)


model.fit([X_train1,X_train2], Y_train, epochs = 5, batch_size = 10)


import h5py

model.save_weights('model_weights33.h5')


del X_train1, X_train2


img_1=load_img('../input/cedardataset/signatures/full_forg/forgeries_20_10
.png',target_size=(155,220))

x = img_to_array(img_1)

x=preprocess(x)

l1=[x]

# l2=[x2]

l1=np.asarray(l1)
```

```
# l2=np.asarray(l2)


l1=[x]

# l2=[x2]

l1=np.asarray(l1)


img_1=load_img('../input/cedardataset/signatures/full_org/original_20_10.png',target_size=(155,220))

x = img_to_array(img_1)

x=preprocess(x)


l2=[x]

l2=np.asarray(l2)


pred=model.predict([l1,l2])


pred
```

**3.mongodb.py**

```python
import pymongo

import uuid

from pymongo import MongoClient


client = MongoClient('mongodb://localhost:27017/')

db = client['BankBot']

User=db['User']

print(db.list_collection_names())


class Users:

    def __init__(self) -> None:

        self.client = MongoClient('mongodb://localhost:27017/')

        self.db = client['BankBot']

        self.User=self.db['User']


    def ifuserexists(self,username):

        user=self.User.find_one({'username':username})

        print(user)

        if user is None:

            return 1

        else:

            return 0
```

```
def addnewuser(self,username,password,fingerprint="",signature=""):

    new_user={

        "username":username,

        "password":password,

        "fingerprint_Path":fingerprint,

        "signature_path":signature

    }

    if self.User.insert_one(new_user):

        return 1

    return 0


def getuser(self,username):

    if self.ifuserexists(username) == 0:

        return self.User.find_one({'username':username})
```

**4.app.py**

```python
from flask import Flask,render_template,session,redirect,url_for

import numpy as np

from flask import jsonify,request

import joblib

import pymongo

import hashlib

import os


from Scripts.mongodb import Users

from Scripts.get_sim import similarity_orb

from Scripts.check_model import GetModel




model = GetModel()

user=Users()




UPLOAD_FOLDER_FIN='D:\mdp_2\Images\Fingerprint'

UPLOAD_FOLDER_SIG='D:\mdp_2\Images\Signatures'


import json


app=Flask(__name__)
```

```
app.config["SESSION_PERMANENT"] = False
app.config["SESSION_TYPE"] = "filesystem"
app.config['SECRET_KEY'] = 'GDtfDCFYjD'
app.config['UPLOAD_FOLDER_FIN']=UPLOAD_FOLDER_FIN
app.config['UPLOAD_FOLDER_SIG']=UPLOAD_FOLDER_SIG



# landing view

@app.route('/')
def landing():

    return render_template('landing.html')



# view for login
@app.route('/login',methods=['GET','POST'])
def login():
    if session['username'] is not None:
        return redirect(url_for('home'))

    if(request.method == 'POST'):
```

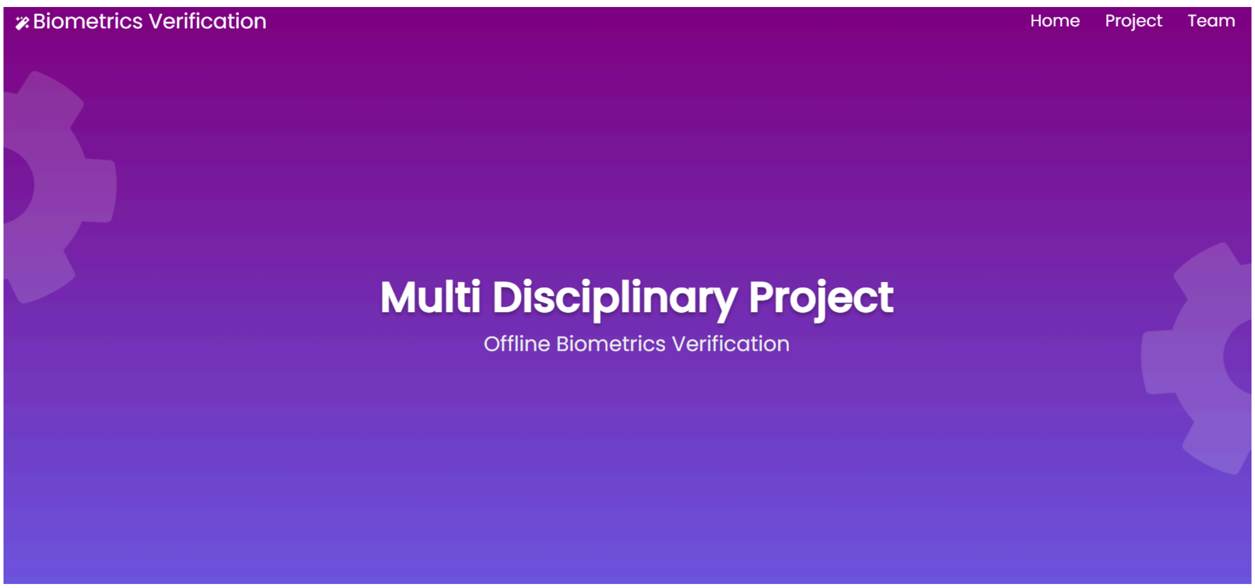
```
        username = request.form.get("username","Anonimous")

        password=request.form.get("password","12345678")

        user_data=user.getuser(username)

        if user_data is None:

            return "Something is wrong"

        if password==user_data['password']:

            session['username']=user_data['username']

            return redirect(url_for('home'))


    else:

        return render_template('login.html')


#view for sign up


@app.route('/signup',methods=['GET','POST'])

def signup():

    if(request.method == 'POST'):

        username = request.form.get("username","Anonimous")

        password=request.form.get("password","12345678")

        fingerprint=request.files["fingerprint"]

        signature=request.files["signature"]


        if(user.ifuserexists(username) == 0):
```

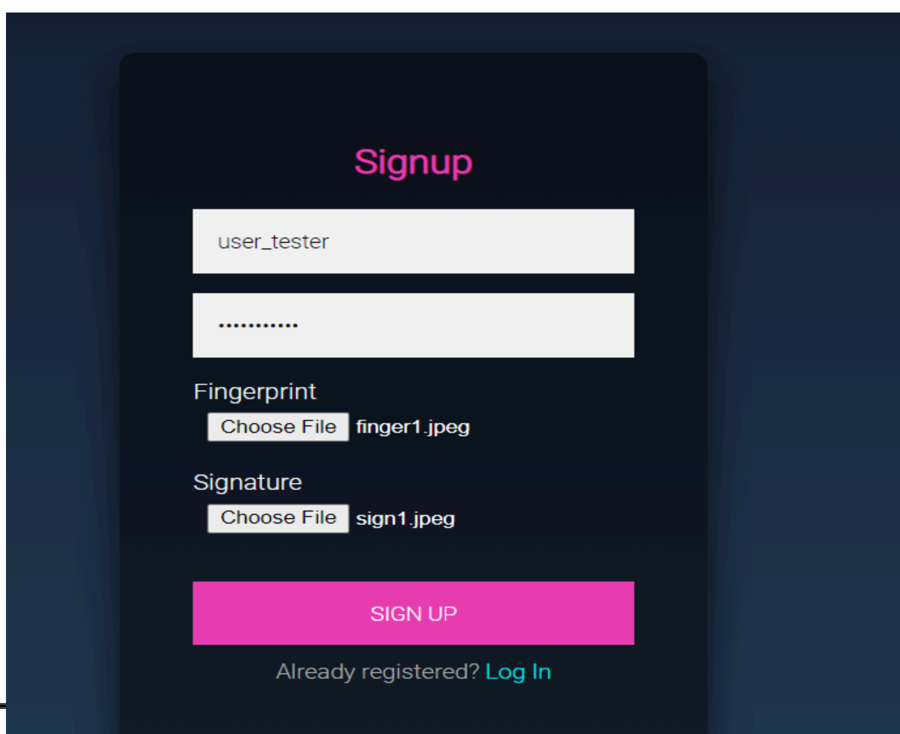
```
            return "User exists"

             fingerprint_path=os.path.join(app.config['UPLOAD_FOLDER_FIN'],
fingerprint.filename)

               signature_path=os.path.join(app.config['UPLOAD_FOLDER_SIG'],
signature.filename)



if(user.addnewuser(username,password,fingerprint_path,signature_path)    ==
0):

            return "cannot add user"

        fingerprint.save(fingerprint_path)

        signature.save(signature_path)

        session['username']=username


        return redirect(url_for('home'))

        return render_template('signup.html')

    return render_template('signup.html')


#view for home

@app.route('/home',methods=['GET','POST'])

def home():

    if request.method=='POST':

        print("FFFF")

        return redirect(url_for('authenticate'))

    return render_template('home.html')
```

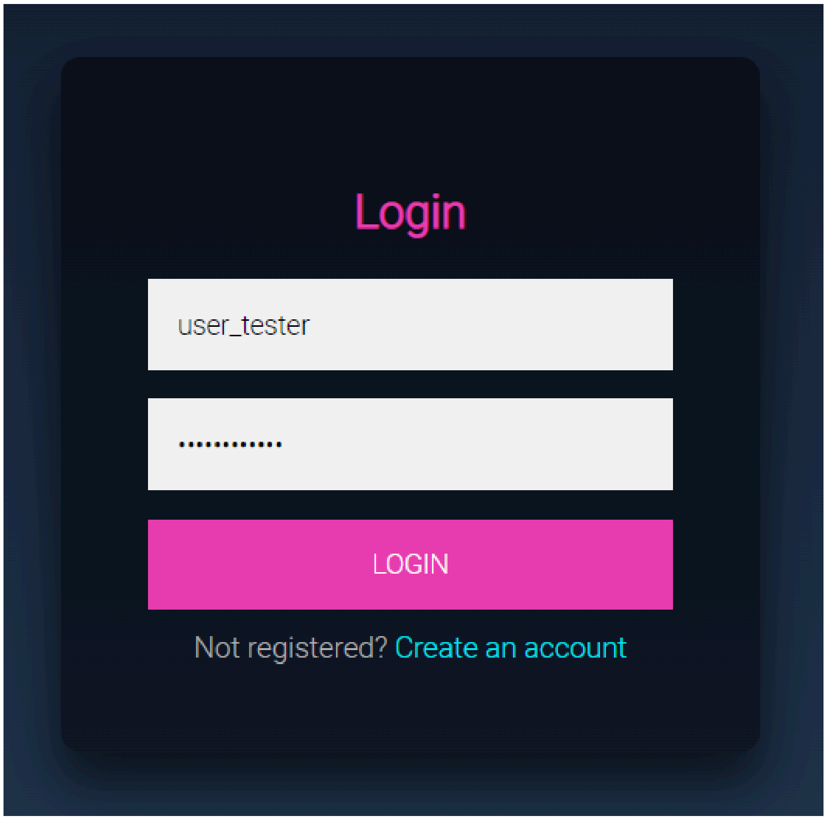
```
# view for logout
@app.route('/logout')

def logout():
    session['username']=None
    return redirect(url_for('landing'))

@app.route('/authenticate',methods=['GET','POST'])
def authenticate():
    if request.method=='POST':
        fingerprint = request.files["fingerprint"]
        signature = request.files["signature"]

temp_fingerprint_path=os.path.join('D:/mdp_2/Images/temproary/',fingerprin
t.filename)

temp_sign_path=os.path.join('D:/mdp_2/Images/temproary/',signature.filenam
e)
        fingerprint.save(temp_fingerprint_path)
        signature.save(temp_sign_path)
        print(session['username'])
        username=session['username']
        user_details=user.getuser(username=username)
```

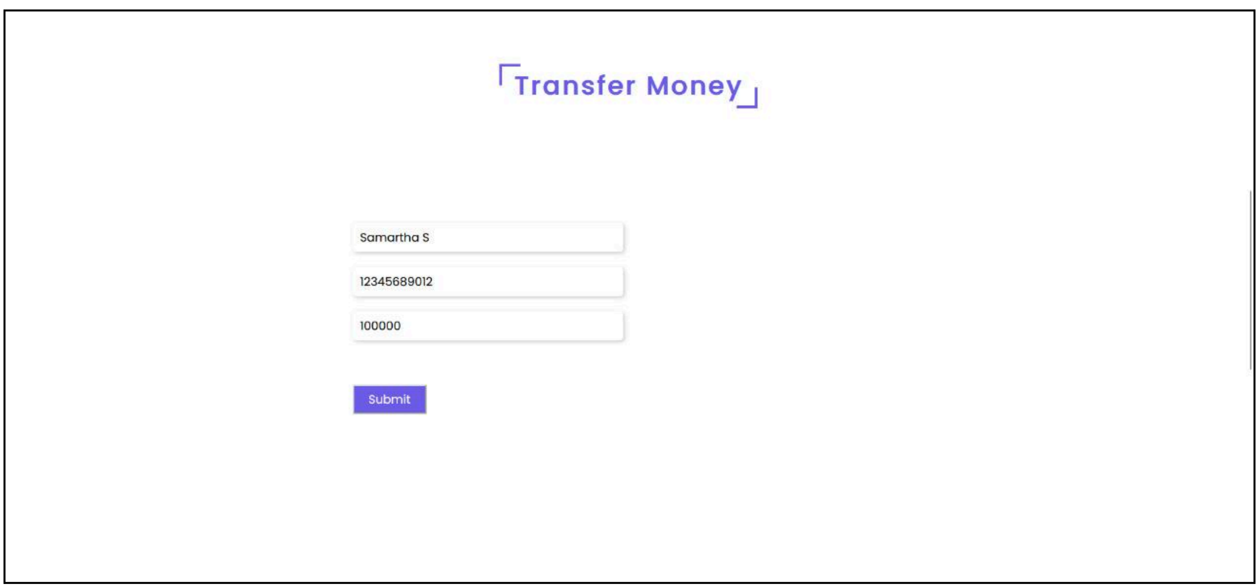
```
                                                                fin_mathces       =
similarity_orb(user_details['fingerprint_Path'],temp_fingerprint_path)

                                                               sign_mathces       =
model.predict_model(user_details['signature_path'],temp_sign_path)

        print(sign_mathces)

        if sign_mathces == 1 and fin_mathces ==1 :


            return "<h2>Money is debited from your account</h2>"


        else:

            return "<h2>Invalid biometrics</h2>"


        return render_template('fileinput.html')

    return render_template('fileinput.html')


if __name__=='__main__':

    app.run(port=5000,debug=True)
```

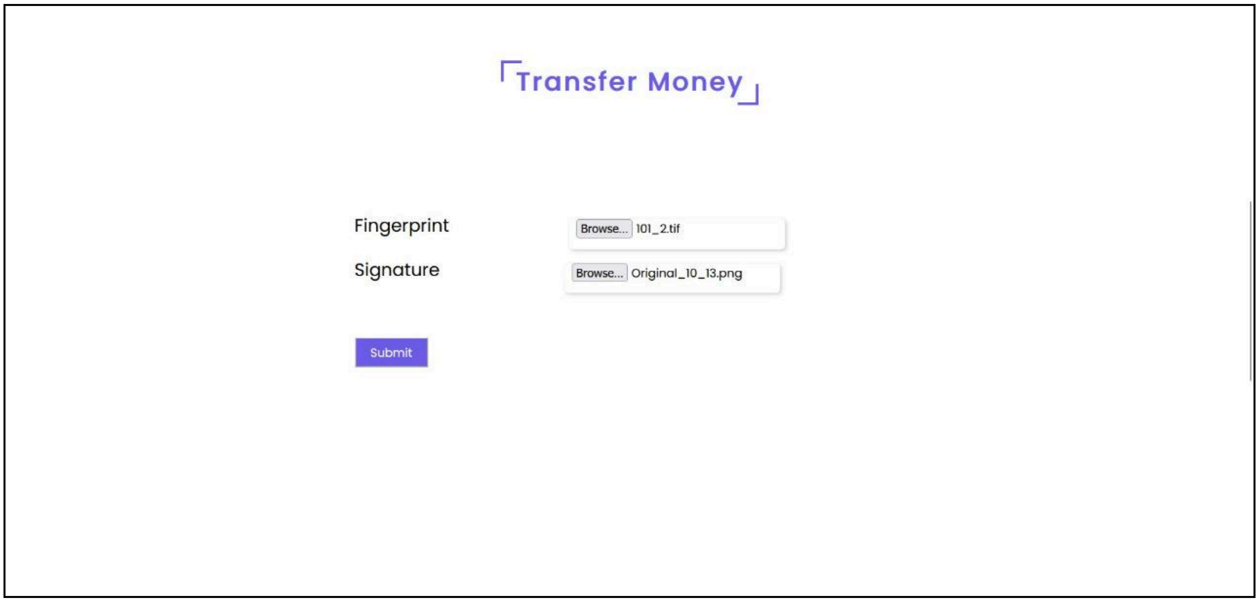
# 5. Results


Biometrics Verification
Home
Project
Team
Multi Disciplinary Project
Offline Biometrics Verification


Signup
user_tester
Fingerprint
Choose File
finger1.jpeg
Signature
Choose File
sign1.jpeg
SIGN UP
Already registered? Log In


Login
user_tester
LOGIN
Not registered? Create an account


Transfer Money
Samartha S
12345689012
100000
Submit

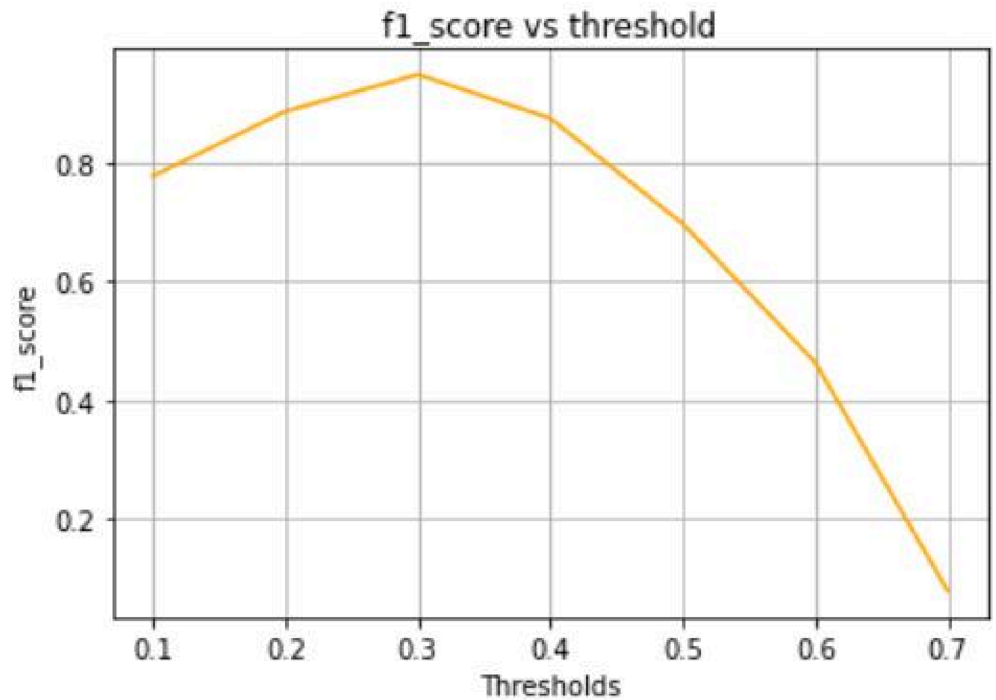


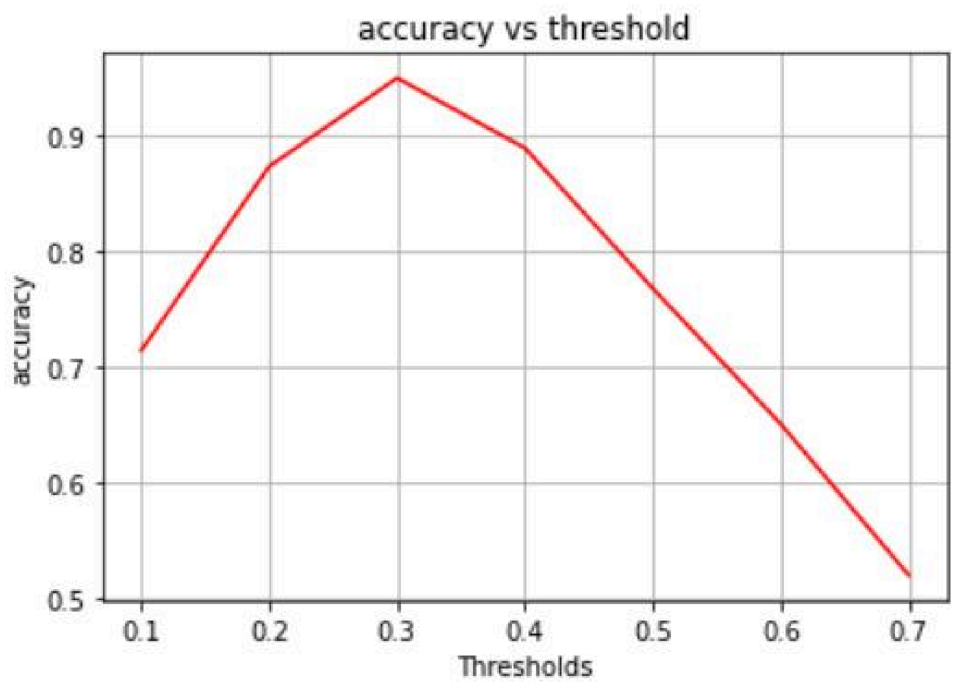

```
Fingerprint matches.
1/1 [==============================] - 0s 135ms/step
1
127.0.0.1 - - [27/Jul/2022 19:34:09] "POST /authenticate HTTP/1.1" 200 -
```

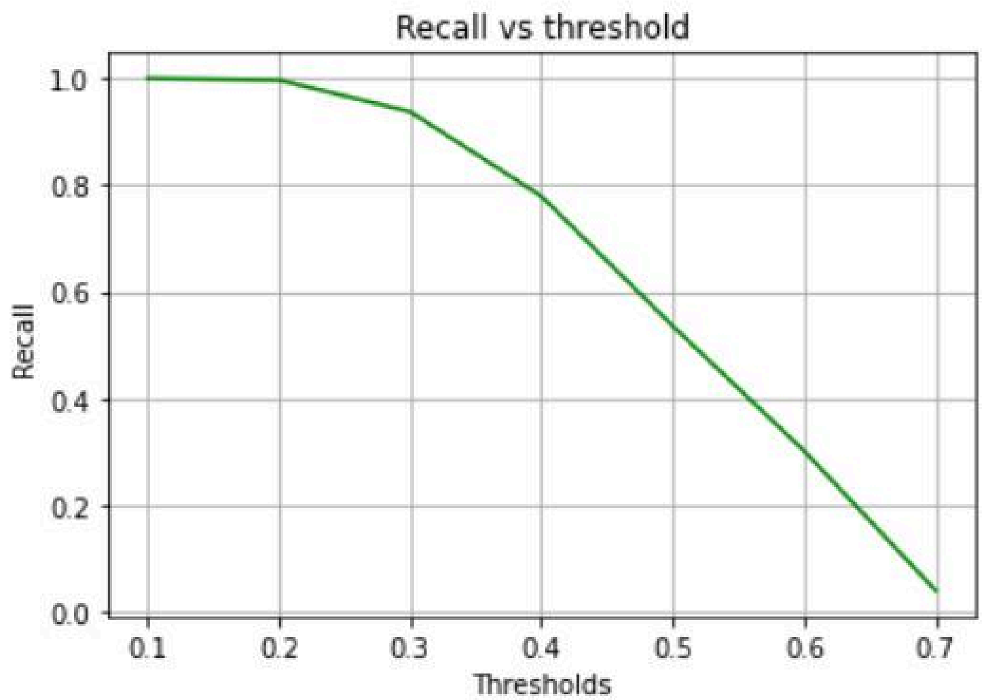
**Money is debited from your account**

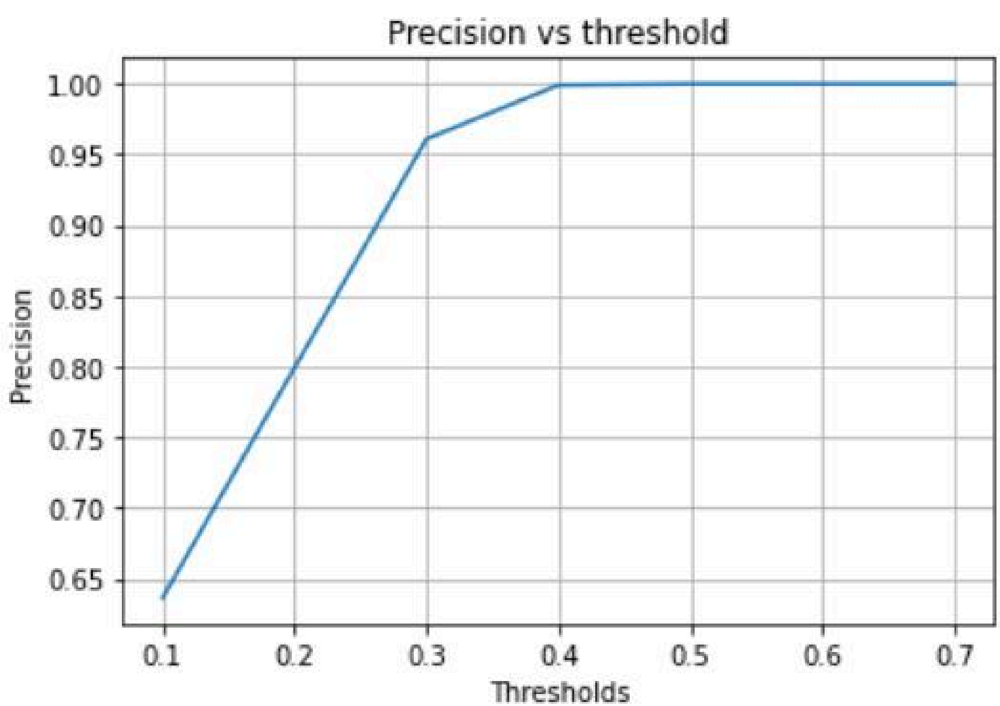
```
_by_count=0. The caller indicates that this is not a failure, but this may mean that there could b
1/1 [==============================] - 18s 18s/step
0
127.0.0.1 - - [27/Jul/2022 19:31:10] "POST /authenticate HTTP/1.1" 200 -
```

**Invalid biometrics**


f1_score vs threshold
f1_score
0.2
0.4
0.6
0.8
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Thresholds


accuracy vs threshold
accuracy
0.5
0.6
0.7
0.8
0.9
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Thresholds


Recall vs threshold
Recall
Thresholds
1.0
0.8
0.6
0.4
0.2
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7


Precision vs threshold
Precision
Thresholds
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.1
0.2
0.3
0.4
0.5
0.6
0.7

# 6. Conclusion and Future Work

We have successfully implemented a mechanism that can verify a user using his/her signature and fingerprint. Our proposed model will provide better results than the current process of manual testing. With a few improvements and modifications, our system can replace the current manual process and automate the entire process.

**Future Work:**

1. Extending this project to e-KYC
2. Extending the concept to online depositing of cheque

# 7. References


1. Ranganathan, G., 2021. A study to find facts behind preprocessing on deep learning algorithms. *Journal of Innovative Image Processing (JIIP)*, *3*(01), pp.66-74.
2. Lake, B., Salakhutdinov, R., Gross, J. and Tenenbaum, J., 2011. One shot learning of simple visual concepts. In *Proceedings of the annual meeting of the cognitive science society* (Vol. 33, No. 33).
3. Woodward, M. and Finn, C., 2017. Active one-shot learning. *arXiv preprint arXiv:1702.06559*.
4. Koch, G., Zemel, R. and Salakhutdinov, R., 2015, July. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop* (Vol. 2, p. 0).
5. Wang, Q., Teng, Z., Xing, J., Gao, J., Hu, W. and Maybank, S., 2018. Learning attentions: residual attentional siamese network for high performance online visual tracking. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4854-4863).
6. He, A., Luo, C., Tian, X. and Zeng, W., 2018. A twofold siamese network for real-time object tracking. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4834-4843).
7. Maind, S.B. and Wankar, P., 2014. Research paper on basic of artificial neural network. *International Journal on Recent and Innovation Trends in Computing and Communication*, *2*(1), pp.96-100.
8. O'Shea, K. and Nash, R., 2015. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*.
9. Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J. and Chen, T., 2018. Recent advances in convolutional neural networks. *Pattern recognition*, *77*, pp.354-377.
10. Tolosana, R., Vera-Rodriguez, R., Fierrez, J. and Ortega-Garcia, J., 2021. DeepSign: Deep on-line signature verification. *IEEE Transactions on Biometrics, Behavior, and Identity Science*, *3*(2), pp.229-239.
11. Galbally, J., Plamondon, R., Fierrez, J. and Ortega-Garcia, J., 2012. Synthetic on-line signature generation. Part I: Methodology and algorithms. *Pattern Recognition*, *45*(7), pp.2610-2621.
12. Galbally, J., Fierrez, J., Ortega-Garcia, J. and Plamondon, R., 2012. Synthetic on-line signature generation. Part II: Experimental validation. *Pattern Recognition*, *45*(7), pp.2622-2632.
13. Rajput, G.G. and Patil, P., 2017. Writer independent offline signature recognition based upon HOGs features. *Int J Electr Eng*, *9*(1), pp.59-67.

14. McCabe, A. and Trevathan, J., 2008, December. Markov model-based handwritten signature verification. In *2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing* (Vol. 2, pp. 173-179). IEEE.
15. Nguyen, V., Kawazoe, Y., Wakabayashi, T., Pal, U. and Blumenstein, M., 2010, November. Performance analysis of the gradient feature and the modified direction feature for off-line signature verification. In *2010 12th International Conference on Frontiers in Handwriting Recognition* (pp. 303-307). IEEE.
16. Shaikh, M.A., Duan, T., Chauhan, M. and Srihari, S.N., 2020, September. Attention based writer independent verification. In *2020 17th International Conference on Frontiers in Handwriting Recognition (ICFHR)* (pp. 373-379). IEEE.
17. Dey, S., Dutta, A., Toledo, J.I., Ghosh, S.K., Lladós, J. and Pal, U., 2017. Signet: Convolutional siamese network for writer independent offline signature verification. *arXiv preprint arXiv:1707.02131*.
18. Hafemann, L.G., Sabourin, R. and Oliveira, L.S., 2017, November. Offline handwritten signature verification—literature review. In *2017 seventh international conference on image processing theory, tools and applications (IPTA)* (pp. 1-8). IEEE.
19. Pansare, A. and Bhatia, S., 2012. Handwritten signature verification using neural network. *International Journal of Applied Information Systems*, *1*(2), pp.44-49.
20. Militello, C., Rundo, L., Vitabile, S. and Conti, V., 2021. Fingerprint classification based on deep learning approaches: Experimental findings and comparisons. *Symmetry*, *13*(5), p.750.
21. Sagayam, K.M., Ponraj, D.N., Winston, J., Yaspy, J.C., Jeba, D.E. and Clara, A., 2019. Authentication of biometric system using fingerprint recognition with euclidean distance and neural network classifier. *Int. J. Innov. Technol. Explor. Eng*, *8*(4), pp.766-771.