Exisitng method-Multimodal biometric authentication based on Multi- support vector neural network with deep belief neural network (Multi-SVNN -DBN)

```
import os
import numpy as np
import pandas as pd
import cv2
import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')

from sklearn.model_selection import train_test_split
import itertools

from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau
from keras.models import Sequential, Model
from keras.layers import Dense, Activation, Flatten, Dropout, concatenate, Input, Conv2D, MaxPooling2D
from keras.optimizers import Adam, Adadelta
from keras.layers.advanced_activations import LeakyReLU
from keras.utils.np_utils import to_categorical
Using TensorFlow backend.
Load Data
train_dir = '../input/plant-seedlings-classification/train'
test_dir = '../input/plant-seedlings-classification/test'
sample_submission = pd.read_csv('../input/plant-seedlings-classification/sample_submission.csv')
Different Species
SPECIES = ['Black-grass', 'Charlock', 'Cleavers', 'Common Chickweed', 'Common wheat', 'Fat Hen',
              'Loose Silky-bent', 'Maize', 'Scentless Mayweed', 'Shepherds Purse',
              'Small-flowered Cranesbill', 'Sugar beet']

for species in SPECIES:
    print('{} {} images'.format(species, len(os.listdir(os.path.join(train_dir, species)))))

Black-grass 263 images
Charlock 390 images
Cleavers 287 images
Common Chickweed 611 images
Common wheat 221 images
Fat Hen 475 images
Loose Silky-bent 654 images
Maize 221 images
Scentless Mayweed 516 images
Shepherds Purse 231 images
Small-flowered Cranesbill 496 images
Sugar beet 385 images
Training Data Files
train = []

for species_num, species in enumerate(SPECIES):
    for file in os.listdir(os.path.join(train_dir, species)):
        train.append(['../input/plant-seedlings-classification/train/{}/{}'.format(species, file), species_num, species])

train = pd.DataFrame(train, columns=['file', 'species_num', 'species'])

print('Training Data: ',train.shape)
Training Data:  (4750, 3)
Image Pre-processing
def create_mask_for_plant(image):
    image_hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    sensitivity = 35
    lower_hsv = np.array([60 - sensitivity, 100, 50])
    upper_hsv = np.array([60 + sensitivity, 255, 255])

    mask = cv2.inRange(image_hsv, lower_hsv, upper_hsv)
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (11,11))
    mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)

    return mask

def segment_plant(image):
```

```
        mask = create_mask_for_plant(image)
        output = cv2.bitwise_and(image, image, mask = mask)
        return output

    def sharpen_image(image):
        image_blurred = cv2.GaussianBlur(image, (0, 0), 3)
        image_sharp = cv2.addWeighted(image, 1.5, image_blurred, -0.5, 0)
        return image_sharp
    Loading Traing Data
    %%time

    x_train = []

    for i in range(len(train)):
        img = cv2.imread(train['file'][i])
        img = cv2.resize(img,dsize=(256,256))
        img_stack = segment_plant(img)
        img_stack = sharpen_image(img_stack)
        img_stack = cv2.cvtColor( img_stack, cv2.COLOR_RGB2GRAY )
        img_stack = np.reshape(img_stack,(256,256,1))
        x_train.append(np.concatenate((np.array(img),np.array(img_stack)),axis=2))

    x_train = np.array(x_train)
    CPU times: user 58.5 s, sys: 7.65 s, total: 1min 6s
    Wall time: 1min 13s
    Sample Images
    # Input image
    Input_image = cv2.imread(train['file'][len(train)-1])

    plt.imshow(Input_image)
    plt.title('Input image, Shape: '+str(Input_image.shape))
    plt.show()

    # Resized image
    plt.imshow(img)
    plt.title('Resized image, Shape: '+str(img.shape))
    plt.show()

    # Processed image to Stack
    plt.imshow(np.reshape(img_stack,(256,256)))
    plt.title('Processed image, Shape: '+str(img_stack.shape))
    plt.show()



    One-hot Encoding
    labels = train['species_num']
    labels = to_categorical(labels, num_classes = len(SPECIES))
    CV-Partition
    x_train, x_val, y_train, y_val = train_test_split(x_train, labels, test_size = 0.1, random_state=10)
    Input Shape
    input_shape = x_train[1].shape
    print('Input Shape is :', input_shape)
    Input Shape is : (256, 256, 4)
    Architecture
    def fire_incept(x, fire=16, intercept=64):
        x = Conv2D(fire, (5,5), strides=(2,2))(x)
        x = LeakyReLU(alpha=0.15)(x)

        left = Conv2D(intercept, (3,3), padding='same')(x)
        left = LeakyReLU(alpha=0.15)(left)

        right = Conv2D(intercept, (5,5), padding='same')(x)
        right = LeakyReLU(alpha=0.15)(right)

        x = concatenate([left, right], axis=3)
        return x

    def fire_squeeze(x, fire=16, intercept=64):
        x = Conv2D(fire, (1,1))(x)
        x = LeakyReLU(alpha=0.15)(x)

        left = Conv2D(intercept, (1,1))(x)
        left = LeakyReLU(alpha=0.15)(left)

        right = Conv2D(intercept, (3,3), padding='same')(x)
        right = LeakyReLU(alpha=0.15)(right)

        x = concatenate([left, right], axis=3)
        return x

    image_input=Input(shape=input_shape)
```

```
x = fire_incept((image_input), fire=16, intercept=16)

x = fire_incept(x, fire=32, intercept=32)
x = fire_squeeze(x, fire=32, intercept=32)

x = fire_incept(x, fire=64, intercept=64)
x = fire_squeeze(x, fire=64, intercept=64)

x = fire_incept(x, fire=64, intercept=64)
x = fire_squeeze(x, fire=64, intercept=64)

x = Conv2D(64, (3,3))(x)
x = LeakyReLU(alpha=0.1)(x)

x = Flatten()(x)

x = Dense(512)(x)
x = LeakyReLU(alpha=0.1)(x)
x = Dropout(0.1)(x)

out = Dense(len(SPECIES), activation='softmax')(x)

model_new = Model(image_input, out)
model_new.summary()
```

```
pip install Dlib
```

```
Requirement already satisfied: Dlib in /usr/local/lib/python3.7/dist-packages (19.18.0)
```

```
pip install face_recognition
```

```
Requirement already satisfied: face_recognition in /usr/local/lib/python3.7/dist-packages (1.3.0)
Requirement already satisfied: dlib>=19.7 in /usr/local/lib/python3.7/dist-packages (from face_recognition) (19.18.0)
Requirement already satisfied: face-recognition-models>=0.3.0 in /usr/local/lib/python3.7/dist-packages (from face_recognition) (0.
Requirement already satisfied: Click>=6.0 in /usr/local/lib/python3.7/dist-packages (from face_recognition) (7.1.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from face_recognition) (1.21.5)
Requirement already satisfied: Pillow in /usr/local/lib/python3.7/dist-packages (from face_recognition) (7.1.2)
```

```
pip install opencv-python
```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.7/dist-packages (4.1.2.30)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from opencv-python) (1.21.5)
```

```
import dlib
import csv
```

Proposed method-Recalling-Enhanced Recurrent Neural Network (RE-RNN) and Graph-Embedded Convolutional Neural Network based Multimodal biometric recognition

```
import cv2
import numpy as np
import face_recognition
from google.colab.patches import cv2_imshow
import os
```

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

Load face image

1 frames

```
img = face_recognition.load_image_file('/content/drive/MyDrive/dataset/student_images/2P002M051.jpg')
cv2_imshow(img)
cv2.waitKey(0)
```

```
27
```

```
import numpy as np
import cv2

face = face_recognition.face_locations(img)[0]
copy = img.copy()
#------------------Drawing the Rectangle-------------------------
cv2.rectangle(copy, (face[3], face[0]),(face[1], face[2]), (255,0,255), 2)
cv2_imshow(copy)
cv2.waitKey(0)

img= cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

star = cv2.xfeatures2d.StarDetector_create()

brief = cv2.xfeatures2d.BriefDescriptorExtractor_create()

kp = star.detect(img,None)

kp, des = brief.compute(img, kp)
inputimage = face_recognition.face_encodings(img)[0]
print(inputimage)
```

Load voice data

```
import numpy as np
import scipy
from scipy.io import wavfile
import scipy.fftpack as fft
from scipy.signal import get_window
import IPython.display as ipd
import matplotlib.pyplot as plt
import numpy as np
from python_speech_features import mfcc, logfbank
```

```
pip install python_speech_features
```

```
frequency_sampling, audio_signal=wavfile.read("/content/drive/MyDrive/dataset/voice/1P001E300.WAV")
```

```
audio_signal = audio_signal[:15000]

features_mfcc = mfcc(audio_signal, frequency_sampling)

print('\nMFCC:\nNumber of windows =', features_mfcc.shape[0])
print('Length of each feature =', features_mfcc.shape[1])

features_mfcc = features_mfcc.T
plt.matshow(features_mfcc)
plt.title('MFCC')

filterbank_features = logfbank(audio_signal, frequency_sampling)

print('\nFilter bank:\nNumber of windows =', filterbank_features.shape[0])
print('Length of each feature =', filterbank_features.shape[1])

filterbank_features = filterbank_features.T
plt.matshow(filterbank_features)
plt.title('Filter bank')
plt.show()
print(type(features_mfcc))
```

```
b=np.sum(features_mfcc)
print(b)
```

Load left hand and right hand images

```
import skimage
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
from skimage import feature
import os
import cv2
import json
from skimage import io
filename = os.path.join(os.getcwd(),'/content/drive/MyDrive/dataset/hand/left/0P002L1.jpg')
bird =io.imread(filename)
im = cv2.cvtColor(bird, cv2.COLOR_BGR2GRAY)
staL= feature.canny(im,sigma=1)
plt.imshow(staL)
from PIL import Image
im = Image.fromarray(staL)
im.save("your_fileL.jpeg")
def read_this(image_file, gray_scale=False):
    image_src =cv2.imread('/content/your_fileL.jpeg')
    if gray_scale:
        image_src = cv2.cvtColor(image_src, cv2.COLOR_BGR2GRAY)
        print(image_src)
    else:
        image_src = cv2.cvtColor(image_src, cv2.COLOR_BGR2RGB)
        print(image_src)
    return image_src
def binarize_lib(image_file, thresh_val=127, with_plot=False, gray_scale=False):
    image_src = read_this(image_file=image_file, gray_scale=gray_scale)
    th, image_b = cv2.threshold(src=image_src, thresh=thresh_val, maxval=255, type=cv2.THRESH_BINARY)

    if with_plot:
        cmap_val = None if not gray_scale else 'gray'
        fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(20, 6))

        ax1.axis("off")
        ax1.title.set_text('Multiresolution')

        ax2.axis("off")
        ax2.title.set_text("Binarized")
        ax1.imshow(image_src, cmap=cmap_val)
        ax2.imshow(image_b, cmap=cmap_val)

        return True
    return image_b
binarize_lib(image_file='your_file.jpeg', with_plot=True, gray_scale=True)



filename = os.path.join(os.getcwd(),'/content/drive/MyDrive/dataset/hand/right/0P001R1.jpg')
bird =io.imread(filename)
im = cv2.cvtColor(bird, cv2.COLOR_BGR2GRAY)
staR= feature.canny(im,sigma=1)
plt.imshow(staR)

from PIL import Image
im = Image.fromarray(staR)
im.save("your_fileR.jpeg")
def read_this(image_file, gray_scale=False):
    image_src =cv2.imread('/content/your_fileR.jpeg')
    if gray_scale:
        image_src = cv2.cvtColor(image_src, cv2.COLOR_BGR2GRAY)
        print(image_src)
    else:
        image_src = cv2.cvtColor(image_src, cv2.COLOR_BGR2RGB)
        print(image_src)
    return image_src
def binarize_lib(image_file, thresh_val=127, with_plot=False, gray_scale=False):
    image_src = read_this(image_file=image_file, gray_scale=gray_scale)
    th, image_b = cv2.threshold(src=image_src, thresh=thresh_val, maxval=255, type=cv2.THRESH_BINARY)
    if with_plot:
        cmap_val = None if not gray_scale else 'gray'
        fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(20, 6))

        ax1.axis("off")
        ax1.title.set_text('Multiresolution')
```

```
            ax2.axis("off")
            ax2.title.set_text("Binarized")
            ax1.imshow(image_src, cmap=cmap_val)
            ax2.imshow(image_b, cmap=cmap_val)

            return True
        return image_b
    binarize_lib(image_file='your_file.jpeg', with_plot=True, gray_scale=True)

    out1=np.concatenate((staL,staR), axis=0)
```

Bicom-MASK

```
    pip install torch==1.9.1


    from torch.utils.data import Dataset, DataLoader, Subset
    import cv2
    from PIL import Image
    import pickle

    class FusionDataset(Dataset):

        def __init__(self,df,inputs_cam,masks_cam,inputs_flau,masks_flau,transform=None,mode='train'):
            self.df = df
            self.transform=transform
            self.mode=mode
            self.inputs_cam=inputs_cam
            self.masks_cam=masks_cam
            self.inputs_flau=inputs_flau
            self.masks_flau=masks_flau

        def __len__(self):
            return len(self.df)

        def __getitem__(self,idx):

            im_path = self.df.iloc[idx]['image_path']
            img = cv2.imread(im_path)
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            img=Image.fromarray(img)
            if self.transform is not None:
                img = self.transform(img)
            img=img.cuda()
            input_id_cam=self.inputs_cam[idx].cuda()
            input_mask_cam=self.masks_cam[idx].cuda()
            input_id_flau=self.inputs_flau[idx].cuda()
            input_mask_flau=self.masks_flau[idx].cuda()

            if self.mode=='test':
                return img,input_id_cam,input_mask_cam,input_id_flau,input_mask_flau
            else:

                labels = torch.tensor(self.df.iloc[idx]['labels']).cuda()

                return img,input_id_cam,input_mask_cam,input_id_flau,input_mask_flau,labels
    pickle.dump(out1, open('file_name1.pickle', 'wb'))
    pickle.dump(inputimage, open('file_name2.pickle', 'wb'))
    pickle.dump(features_mfcc, open('file_name3.pickle', 'wb'))
    class vector_fusion():

        def __init__(self):
            super(vector_fusion, self).__init__()
            self.img_model = SEResnext50_32x4d(pretrained=None)
            self.img_model.load_state_dict(torch.load('../input/seresnext2048/best_model.pt'))
            self.img_model.l0=Identity()
            for params in self.img_model.parameters():
                params.requires_grad=False

            self.cam_model= vec_output_CamembertForSequenceClassification.from_pretrained(
        'camembert-base', # Use the 12-layer BERT model, with an uncased vocab.
        num_labels = len(Preprocess.dict_code_to_id), # The number of output labels--2 for binary classification.

        output_attentions = False,
        output_hidden_states = False,)

            cam_model_path = '../input/camembert-vec-256m768-10ep/best_model.pt'
            checkpoint = torch.load(cam_model_path)
            # model = checkpoint['model']
```

```
        self.cam_model.load_state_dict(checkpoint)
        for param in self.cam_model.parameters():
            param.requires_grad=False
        self.cam_model.out_proj=Identity()

        self.flau_model=vec_output_FlaubertForSequenceClassification.from_pretrained(
        'flaubert/flaubert_base_cased',
        num_labels = len(Preprocess.dict_code_to_id),
        output_attentions = False,
        output_hidden_states = False,)
        flau_model_path='../input/flaubert-8933/best_model.pt'
        checkpoint = torch.load(flau_model_path)
        self.flau_model.load_state_dict(checkpoint)
        for param in self.flau_model.parameters():
            param.requires_grad=False
        self.flau_model.classifier=Identity()


        self.reduce_dim=nn.Conv1d(in_channels = 2048 , out_channels = 768 , kernel_size= 1)
        self.reduce_dim2=nn.Conv1d(in_channels = 768 , out_channels = 1 , kernel_size= 1)
        self.out=nn.Linear(768*3, 27)



    def forward(self,img,input_id_cam,input_mask_cam,input_id_flau,input_mask_flau):

        cam_emb,vec1 =self.cam_model(input_id_cam,
                     token_type_ids=None,
                     attention_mask=input_mask_cam)
        flau_emb,vec2 =self.flau_model(input_id_flau,
                     token_type_ids=None,
                     attention_mask=input_mask_flau)

        #Projecting the image embedding to lower dimension
        img_emb=self.img_model(img)
        img_emb=img_emb.view(img_emb.shape[0],img_emb.shape[1],1)
        img_emb=self.reduce_dim(img_emb)
        img_emb=img_emb.view(img_emb.shape[0],img_emb.shape[1]) ###### bs * 768

        fuse= torch.cat([img_emb,cam_emb[0],flau_emb[0]],axis=1)

        logits=self.out(fuse)
        return logits
```

RERNN

load face image dataset for training

```
path='/content/drive/MyDrive/dataset/student_images'
images = []
classNames =[]
myList = os.listdir(path)
print(myList)
for c1 in myList:
    curImg = cv2.imread(f'{path}/{c1}')
    images.append(curImg)
    classNames.append(os.path.splitext(c1)[0])
print(classNames)



def findEncodings(images):
    encodeList = []
    for img in images:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        encode = face_recognition.face_encodings(img)[0]
        encodeList.append(encode)
    return encodeList
# Find encodings of training images
knownEncodes = findEncodings(images)
```

Load voice signal dataset for training

```
path = '/content/drive/MyDrive/dataset/VOICE'
```

```
pip install python_speech_features
```

```
from scipy.io import wavfile
from python_speech_features import mfcc, logfbank
frequency_sampling, audio_signal=wavfile.read("/content/drive/MyDrive/dataset/voice/1P001E100.WAV")
frequency_sampling1, audio_signal1=wavfile.read("/content/drive/MyDrive/dataset/voice/1P001E200.WAV")
frequency_sampling2, audio_signal2=wavfile.read("/content/drive/MyDrive/dataset/voice/1P001E300.WAV")
audio_signal = audio_signal[:15000]
features_mfcc = mfcc(audio_signal, frequency_sampling)
features_mfcc = features_mfcc.T
audio_signal1 = audio_signal1[:15000]
features_mfcc1 = mfcc(audio_signal1, frequency_sampling1)
features_mfcc1 = features_mfcc1.T
audio_signal2 = audio_signal2[:15000]
features_mfcc2 = mfcc(audio_signal2, frequency_sampling1)
features_mfcc2 = features_mfcc2.T
out=np.concatenate((features_mfcc, features_mfcc1,features_mfcc2), axis=0)
np.save('voice',out)
```

load left hand and right hand images for training

```
path = '/content/drive/MyDrive/dataset/hand/left'
```

```
imagesl = []
classNamesl =[]
myList = os.listdir(path)
print(myList)
for c1 in myList:
    curImg = cv2.imread(f'{path}/{c1}')
    images.append(curImg)
    classNamesl.append(os.path.splitext(c1)[0])
print(classNamesl)
```

```
path = '/content/drive/MyDrive/dataset/hand/right'
```

```
imagesr = []
classNamesr =[]
myList = os.listdir(path)
print(myList)
for c1 in myList:
    curImg = cv2.imread(f'{path}/{c1}')
    images.append(curImg)
    classNamesr.append(os.path.splitext(c1)[0])
print(classNamesr)
```

```
class RNN(object):

    def __init__(self, input_dim, hidden_dim, output_dim, depth, lr=0.002):
        self.lr = lr
        self.depth = depth
        self.input_dim = input_dim
        self.hidden_dim = hidden_dim
        self.output_dim = output_dim
        self.U = xavier_init(input_dim, hidden_dim, fc=True)
        self.W = xavier_init(hidden_dim, hidden_dim, fc=True)
        self.V = xavier_init(hidden_dim, output_dim, fc=True)
        # tmp variable
        self.x = None
        self.H = None
        self.Alpha = None

    def forward_prop(self, x):
        batch_size = x.shape[1]
        self.x = x
        self.H = np.zeros((self.depth, batch_size, self.hidden_dim))
        self.Alpha = np.zeros((self.depth, batch_size, self.hidden_dim))
        h_prev = np.zeros((batch_size, self.hidden_dim))
        sigmoid_output = np.zeros((self.depth, batch_size, self.output_dim))
        for t in range(self.depth):
            self.Alpha[t] = self.x[t]@self.U + h_prev@self.W
            self.H[t] = self.relu(self.Alpha[t])
```

```
            o_t = self.H[t]@self.V
            y_t = self.sigmoid(o_t)
            sigmoid_output[t] = y_t
            h_prev = self.H[t]
        return sigmoid_output

    def backward_prop(self, sigmoid_output, output_label):
        batch_size = output_label.shape[1]
        dU = np.zeros(self.U.shape)
        dW = np.zeros(self.W.shape)
        dV = np.zeros(self.V.shape)
        dH_t_front = np.zeros((batch_size, self.hidden_dim))
        for t in range(self.depth-1, 0, -1):
            dY_t = sigmoid_output[t] - output_label[t]
            dO_t = dY_t * sigmoid_output[t] * (1 - sigmoid_output[t])
            dV += self.H[t].T @ dO_t
            dH_t = dO_t @ self.V.T + dH_t_front
            dAlpha_t = self.relu(self.Alpha[t], dH_t, deriv=True)
            dU += self.x[t].T @ dAlpha_t
            if t > 0:
                dW += self.H[t-1].T @ dAlpha_t
            dH_t_front = dAlpha_t @ self.W.T
        self.U -= self.lr * dU
        self.W -= self.lr * dW
        self.V -= self.lr * dV

    def relu(self, x, front_delta=None, deriv=False):
        if deriv == False:
            return x * (x > 0)
        else:
            back_delta = front_delta * 1. * (x > 0)
            return back_delta

    def sigmoid(self, x):
        return np.where(x >= 0,
                        1 / (1 + np.exp(-x)),
                        np.exp(x) / (1 + np.exp(x)))


def xavier_init(c1, c2, w=1, h=1, fc=False):
    fan_1 = c2 * w * h
    fan_2 = c1 * w * h
    ratio = np.sqrt(6.0 / (fan_1 + fan_2))
    params = ratio * (2 * np.random.random((c1, c2, w, h)) - 1)
    if fc:
        params = params.reshape(c1, c2)
    return params


def generate_dataset(data_size, length, split_ratio):
    X = np.random.uniform(0, 1, (data_size, length, 1))
    Y = np.zeros((data_size, length, 1))
    threshold = length / 2.
    for i in range(data_size):
        prefix_sum = 0
        for j in range(length):
            prefix_sum += X[i][j][0]
            Y[i][j][0] = int(prefix_sum > threshold)
    split_point = int(data_size * split_ratio)
    train_x, test_x = X[:split_point], X[split_point:]
    train_y, test_y = Y[:split_point], Y[split_point:]
    return np.swapaxes(train_x, 0, 1), np.swapaxes(test_x, 0, 1), \
           np.swapaxes(train_y, 0, 1), np.swapaxes(test_y, 0, 1)

def main():
    length = 12
    data_size = 1000
    split_ratio = 0.9
    max_iter = 100
    iters_before_test = 10
    batch_size = 25
    train_x, test_x, train_y, test_y = generate_dataset(data_size, length, split_ratio)
    rnn = RNN(1, 10, 1, length)
    for iters in range(max_iter+1):
        st_idx = int(iters % ((split_ratio * length) / batch_size))
        ed_idx = int(st_idx + batch_size)
        sigmoid_output = rnn.forward_prop(train_x[:, st_idx:ed_idx, :])
        rnn.backward_prop(sigmoid_output, train_y[:, st_idx:ed_idx, :])
        loss = np.sum((sigmoid_output - train_y[:, st_idx:ed_idx, :]) ** 2)
        print(knownEncodes)
        print(features_mfcc)
        print(out1)
```

```
        if iters % iters_before_test == 0:
            sigmoid_output = rnn.forward_prop(test_x)
            predict_label = sigmoid_output > 0.5
            accuracy = float(np.sum(predict_label == test_y.astype(bool))) / test_y.size

if __name__ == '__main__':
    main()


score=[]
score = np.linalg.norm(face_recognition.face_encodings(img)[0])
scores.append(score)
imatches = np.argsort(score)
print(score)


f = open('output.csv','w')
f.close()
from google.colab import files
def mark(name):
    with open('/content/output.csv','r+') as f:
        myDataList = f.readlines()
        nameList =[]
        for line in  myDataList:
            entry = line.split(',')
            nameList.append(entry[0])
        if name not in nameList:

            f.writelines(f'\n{name}')
```

GECNN

```
pip install dgl


pip install torch==1.9.1


import dgl
dataset = dgl.data.CoraGraphDataset()


import dgl
import dgl.function as fn
import torch as th
import torch.nn as nn
import torch.nn.functional as F
from dgl import DGLGraph
from dgl.data import CoraGraphDataset


gcn_msg = fn.copy_u(u='h', out='m')
gcn_reduce = fn.sum(msg='m', out='h')

class GCNLayer(nn.Module):
    def __init__(self, in_feats, out_feats):
        super(GCNLayer, self).__init__()
        self.linear = nn.Linear(in_feats, out_feats)

    def forward(self, g, feature):

        with g.local_scope():
            g.ndata['h'] = feature
            g.update_all(gcn_msg, gcn_reduce)
            h = g.ndata['h']
            return self.linear(h)

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.layer1 = GCNLayer(1433, 16)
        self.layer2 = GCNLayer(16, 7)

    def forward(self, g, features):
        x = F.relu(self.layer1(g, features))
        x = self.layer2(g, x)
        return x
net = Net()
print(net)
```

```
test = img
test = cv2.cvtColor(test, cv2.COLOR_BGR2RGB)
test_encode = face_recognition.face_encodings(test)[0]
matches = face_recognition.compare_faces(knownEncodes, test_encode)
print(matches)
faceDist = face_recognition.face_distance(knownEncodes, test_encode)
matchIndex = np.argmin(faceDist)
print(matchIndex)
if matches[matchIndex]:
  name = classNames[matchIndex].upper().lower()
  mark(name)
with open('/content/output.csv','r') as csvfile:
  reader = csv.reader(csvfile)
  rows = [row for row in reader]
print(rows)
res1 = any("2p001m051" in sublist for sublist in rows)
res2 = any("2p001m052" in sublist for sublist in rows)
res3 = any("2p001m053" in sublist for sublist in rows)
res4 = any("2p001m054" in sublist for sublist in rows)
res5 = any("2p001m055" in sublist for sublist in rows)
re1 = any("2p002m051" in sublist for sublist in rows)
re2 = any("2p002m052" in sublist for sublist in rows)
re3 = any("2p002m053" in sublist for sublist in rows)
re4 = any("2p002m054" in sublist for sublist in rows)
re5 = any("2p002m055" in sublist for sublist in rows)
r1 = any("2p003m051" in sublist for sublist in rows)
r2 = any("2p003m052" in sublist for sublist in rows)
r3 = any("2p003m053" in sublist for sublist in rows)
r4 = any("2p003m054" in sublist for sublist in rows)
r5 = any("2p003m055" in sublist for sublist in rows)
print(re1)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-23-43197c700b18> in <module>()
----> 1 test = img
      2 test = cv2.cvtColor(test, cv2.COLOR_BGR2RGB)
      3 test_encode = face_recognition.face_encodings(test)[0]
      4 matches = face_recognition.compare_faces(knownEncodes, test_encode)
      5 print(matches)

NameError: name 'img' is not defined
```

SEARCH STACK OVERFLOW

voice

```
b=-2166.1170011410927


c=555555;res1=1;re1=0;r1=0;re2=0;re3=0;re4=0;re5=0;r2=0;r3=0;r4=0;r5=0;



def load_cora_data():
    g = dataset[0]
    features = g.ndata['feat']
    labels = g.ndata['label']
    train_mask = g.ndata['train_mask']
    test_mask = g.ndata['test_mask']
    return g, features, labels, train_mask, test_mask


def evaluate(model, g, features, labels, mask):
    model.eval()
    with th.no_grad():
        logits = model(g, features)
        logits = logits[mask]
        labels = labels[mask]
        _, indices = th.max(logits, dim=1)
        correct = th.sum(indices == labels)
        return correct.item() * 1.0 / len(labels)

import time
import numpy as np
g, features, labels, train_mask, test_mask = load_cora_data()
# Add edges between each node and itself to preserve old node representations
g.add_edges(g.nodes(), g.nodes())
optimizer = th.optim.Adam(net.parameters(), lr=1e-2)
dur = []
```

```
for epoch in range(50):
    if epoch >=3:
        t0 = time.time()

    net.train()
    logits = net(g, features)
    logp = F.log_softmax(logits, 1)
    loss = F.nll_loss(logp[train_mask], labels[train_mask])

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if epoch >=3:
        dur.append(time.time() - t0)

    acc = evaluate(net, g, features, labels, test_mask)
    print("Epoch {:05d} | Time(s) {:.4f}".format(
            epoch, loss.item(), acc, np.mean(dur)))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-29-721921b09c6f> in <module>()
     21 import time
     22 import numpy as np
---> 23 g, features, labels, train_mask, test_mask = load_cora_data()
     24 # Add edges between each node and itself to preserve old node representations
     25 g.add_edges(g.nodes(), g.nodes())

<ipython-input-29-721921b09c6f> in load_cora_data()
      1
      2 def load_cora_data():
----> 3     g = dataset[0]
      4     features = g.ndata['feat']
      5     labels = g.ndata['label']

NameError: name 'dataset' is not defined
```

SEARCH STACK OVERFLOW

```
if(((res1==True) or (res2==True) or (res3==True) or (res4==True) or (res5==True))and (b==-1910.2958920878564) and (c==555555)):
  print('person-1 identified')
elif(((re1==True) or (re2==True) or(re3==True) or (re4==True) or (re5==True)) and (b==-2166.1170011410927) and (c==555555)):
  print('person-2 identified')
elif(((r1==True) or (r2==True) or(r3==True) or (r4==True) or (r5==True))and (b==-752.4779019944989) and (c==555555)):
  print('person-3 identified')
else:
  print("verification failed")
```

```
verification failed
```