# CODE

## #Load Necessary packages

import shutil

import os

import cv2

import numpy as np

import matplotlib.pyplot as plt

**# IRIS DETECTION**

def sharpen\_image(image, sigma=1.5, strength=1.5):

blurred = cv2.GaussianBlur(image, (0, 0), sigma)

sharpened = cv2.addWeighted(image, 1.0 + strength, blurred, -strength, 0)

return sharpened

def detect\_and\_circle\_iris(image):

eye\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_eye.xml')

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

eyes = eye\_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=11, minSize=(45, 45), maxSize=(100,100))

for (x, y, w, h) in eyes:

center = (x + w // 2, y + h // 2)

radius = int(0.3 \* max(w, h))

cv2.circle(image, center, radius, (0, 255, 0), 2)

return image, eyes

**# ROI DETECTION**

def detect\_roi(image\_path, eyes\_coordinates):

image = cv2.imread(image\_path)

inter\_ocular\_distance = abs(((eyes\_coordinates[1][0] + eyes\_coordinates[1][2])) - ((eyes\_coordinates[0][0] + eyes\_coordinates[0][2] )))

roi\_width = int(1.62 \* inter\_ocular\_distance)

roi\_height = int(0.5 \* inter\_ocular\_distance)

periocular\_x = eyes\_coordinates[1][0] - int(0.62 \* roi\_width)

periocular\_y = eyes\_coordinates[1][1] - int(0.25 \* roi\_height)

forehead\_x = eyes\_coordinates[1][0] - int(0.62 \* roi\_width)

forehead\_y = eyes\_coordinates[1][1] - int(0.9 \* inter\_ocular\_distance) #- int(0.25 \* roi\_height)

periocular\_roi = image[periocular\_y:periocular\_y + roi\_height, periocular\_x:periocular\_x + roi\_width]

forehead\_roi = image[forehead\_y:forehead\_y + roi\_height, forehead\_x:forehead\_x + roi\_width]

return periocular\_roi, forehead\_roi

**# PRE-PROCESSING**

def gamma\_correction(image, gamma = 0.3):

image\_float32 = image.astype(np.float32) / 255.0

corrected\_image = np.power(image\_float32, gamma)

normalized\_image = cv2.normalize(corrected\_image, None, 0, 255, cv2.NORM\_MINMAX, dtype=cv2.CV\_8U)

return normalized\_image

def dog\_filter(image, sigma1=3, sigma2=1):

blur1 = cv2.GaussianBlur(image, (0, 0), sigmaX=sigma1)

blur2 = cv2.GaussianBlur(image, (0, 0), sigmaX=sigma2)

dog\_filtered = blur1 - blur2

return dog\_filtered

def nonlinear\_mapping(image, tau = 255):

mapped\_image = tau \* np.tanh(image / tau)

return mapped\_image

def preprocess\_image(image\_path, output\_path):

img = cv2.imread(image\_path)

img1 = gamma\_correction(img,0.3)

img2 = dog\_filter(img1, 3, 1)

img3 = nonlinear\_mapping(img2, 255)

cv2.imwrite(output\_path, img3)

**# Import necessary library**

import numpy as np

from sklearn.svm import SVC

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Dense, Flatten, concatenate, Dropout, BatchNormalization

from tensorflow.keras import layers, models

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from sklearn.model\_selection import GridSearchCV

**# VPCNN-Training**

def get\_cnn\_features(model, generator):

features = model.predict(generator)

return features

**#layers**

def model (input\_shape, dropout\_rate=0.5):

model = models.Sequential()

model.add(layers.Conv2D(128, (5, 5), strides=[2, 2], activation='relu', input\_shape=input\_shape))

model.add(layers.MaxPooling2D((2, 2), strides=[1, 1]))

model.add(layers.Conv2D(64, (5, 5), strides=[2, 2], activation='relu'))

model.add(layers.MaxPooling2D((2, 2), strides=[1, 1]))

model.add(layers.Conv2D(64, (5, 5), strides=[2, 2],activation='relu'))

model.add(layers.MaxPooling2D((2, 2), strides=[1, 1]))

model.add(layers.Flatten())

model.add(layers.Dense(256, activation='relu'))

model.add(layers.Dropout(dropout\_rate))

model.add(layers.BatchNormalization())

return model

**#LOAD INPUT DATA**

Input\_size = (128, 233, 1)

im\_shape = (128, 233)

TRAINING\_DIR1 = '/content/drive/MyDrive/RPP/rpp/FSVP-PBP-main/FSVP-PBP-main/train/periocular'

TEST\_DIR1 = '/content/drive/MyDrive/RPP/rpp/FSVP-PBP-main/FSVP-PBP-main/test/periocular'

TRAINING\_DIR2 = '/content/drive/MyDrive/RPP/rpp/FSVP-PBP-main/FSVP-PBP-main/train/forehead'

TEST\_DIR2 = '/content/drive/MyDrive/RPP/rpp/FSVP-PBP-main/FSVP-PBP-main/test/forehead'

BATCH\_SIZE = 50

seed = 20

data\_generator = ImageDataGenerator(rescale=1./255)

cnn\_model\_1 = model(input\_size)

cnn\_model\_2 = model(input\_size)

cnn\_model\_1.add(layers.Dropout(0.5))

cnn\_model\_2.add(layers.Dropout(0.5))

num\_classes = 200

**#combined two networks**

combined\_model = combine\_models(cnn\_model\_1, cnn\_model\_2, num\_classes)

combined\_model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

combined\_model.build((None, 128, 233, 3))

combined\_model.summary()

train1\_generator = data\_generator.flow\_from\_directory(

TRAINING\_DIR1, target\_size=im\_shape, shuffle=False, seed=seed,

class\_mode='categorical', batch\_size=BATCH\_SIZE, subset="training", color\_mode='grayscale'

)

train2\_generator = data\_generator.flow\_from\_directory(

TRAINING\_DIR2, target\_size=im\_shape, shuffle=False, seed=seed,

class\_mode='categorical', batch\_size=BATCH\_SIZE, subset="training", color\_mode='grayscale'

)

train1\_labels = train1\_generator.classes

train2\_labels = train2\_generator.classes

**# Extract features from test data**

test1\_generator = data\_generator.flow\_from\_directory(

TEST\_DIR1, target\_size=im\_shape, shuffle=False, seed=seed,

class\_mode='categorical', batch\_size=BATCH\_SIZE, color\_mode='grayscale'

)

test2\_generator = data\_generator.flow\_from\_directory(

TEST\_DIR2, target\_size=im\_shape, shuffle=False, seed=seed,

class\_mode='categorical', batch\_size=BATCH\_SIZE, color\_mode='grayscale'

)

test1\_labels = test1\_generator.classes

test2\_labels = test2\_generator.classes

**# Concatenate features and labels**

train\_features = np.concatenate([get\_cnn\_features(cnn\_model\_1, train1\_generator), get\_cnn\_features(cnn\_model\_2, train2\_generator)], axis=1)

test\_features = np.concatenate([get\_cnn\_features(cnn\_model\_1, test1\_generator), get\_cnn\_features(cnn\_model\_2, test2\_generator)], axis=1)

**#Hyper-Parameter**

param\_grid = {

'kernel': ['linear', 'rbf', 'poly'],

'C': [0.001, 0.01, 0.1, 1, 10, 100],

'gamma': [0.001, 0.01, 0.1, 1, 'scale', 'auto']

}

**#GrideSearchCV\_SVM**

grid\_search = GridSearchCV(SVC(), param\_grid, cv=3)

grid\_search.fit(train\_features, train1\_labels)

best\_svm\_model = grid\_search.best\_estimator\_

test1\_predictions = best\_svm\_model.predict(test\_features)

accuracy = accuracy\_score(test1\_labels, test1\_predictions)

print(f'Accuracy: {accuracy}')

**# Confusion\_matrix**

from sklearn.metrics import confusion\_matrix

conf\_matrix = confusion\_matrix(test1\_labels, test1\_predictions)

print("Confusion Matrix:")

print(conf\_matrix)

**#Precision, Recall, and F1 Score**

precision = precision\_score(test1\_labels, test1\_predictions, average='weighted')

recall = recall\_score(test1\_labels, test1\_predictions, average='weighted')

f1 = f1\_score(test1\_labels, test1\_predictions, average='weighted')

print(f"Precision: {precision}")

print(f"Recall: {recall}")

print(f"F1 Score: {f1}")

**#FAR, FRR, VR, EER**

def calculate\_rates(confusion\_matrix):

num\_classes = confusion\_matrix.shape[0]

TP = np.diag(confusion\_matrix)

FP = np.sum(confusion\_matrix, axis=1) - TP

FN = np.sum(confusion\_matrix, axis=0) - TP

TN = np.sum(confusion\_matrix) - (TP + FP + FN)

FAR = np.sum(FP) / (np.sum(FP) + np.sum(TN)) # False Acceptance Rate

FRR = np.sum(FN) / (np.sum(FN) + np.sum(TP)) # False Rejection Rate

VR = np.sum(TP) / (np.sum(TP) + np.sum(FN)) # Verification Rate

EER = 0.5 \* (FAR + FRR) # Equal Error Rate

return FAR, FRR, VR, EER

confusion\_matrix = confusion\_matrix(test1\_labels, test1\_predictions)

FAR, FRR, VR, EER = calculate\_rates(confusion\_matrix)

print(f"False Acceptance Rate (FAR): {FAR:.4f}")

print(f"False Rejection Rate (FRR): {FRR:.4f}")

print(f"Verification Rate (VR): {VR:.4f}")

print(f"Equal Error Rate (EER): {EER:.4f}")