README

This project demonstrates how to perform face recognition using deep learning models. It will help you to build a repository of images using an image named sportsteam.png. It will teach you how to crop faces from this image and save to a local folder. It will also allow you to use the faces as input into your deep learning model.

Setup Instructions:

- In your local drive where your Jupiter python project file is located (yourfile.jpynb) create a folder named **Dataset** and a folder named **faces**.
- In the **Dataset** folder copy the following files from your Anaconda folder: (These can be experimented with to help identify faces within your image)
    - "C:\Users\[yourname]\anaconda3\Lib\site-packages\cv2\data\haarcascade_frontalface_alt.xml"
    - "C:\Users\[yourname]\anaconda3\Lib\site-packages\cv2\data\haarcascade_frontalface_alt_tree.xml"
    - "C:\Users\[yourname]\anaconda3\Lib\site-packages\cv2\data\haarcascade_frontalface_alt2.xml"
    - "C:\Users\[yourname]\anaconda3\Lib\site-packages\cv2\data\haarcascade_frontalface_default.xml"
- Open a new file in your Jupyter Network and perform the following Steps.

STEP 1: Data Preprocessing: load your original image and perform the following tasks to create 4 more versions of the original image:

- Convert to Grayscale
- Resize the Image
- Normalize the Image
- Reshape the Image
- Save all versions of the image to your **Dataset** folder
- LIBRARIES REQUIRED: cv2, **numpy np**

STEP2: VISUALIZE SAMPLES – Using **matplotlib.pyplot plt** Plot your saved versions of each of the 5 files and add pixel location ticks on the X and Y axis to aid in plotting crop areas on your folder. You can use the following code sample as an example:

```
      # Visualize the original image

plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title("Original Image")
plt.xlabel("Pixels (X-axis)")
plt.ylabel("Pixels (Y-axis)")
plt.xticks(np.arange(0, image.shape[1], step=image.shape[1] // 32))
plt.yticks(np.arange(0, image.shape[0], step=image.shape[0] // 32))
plt.show()
```

STEP3: DRAW CROP BOXES around the faces and save each Image

- Please note this is a time consuming effort and you will have to adjust pixel locations until you have captured a good region to crop and save to a separate file. An example of the code to use is:

```
image = cv2.imread("Dataset/sportsteam.png")
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

```
•  image
•
•  # pt1 = top left
•  # pt2 = bottom right
•  # color=(0,255,0) means green color of rectangle RGB
•  cv2.circle(img=image, center=(230,158), radius=27, color=(0,255,0), thickness=2) # Top row
   (L-R) - Face 1
•  cv2.circle(img=image, center=(365,148), radius=27, color=(0,255,0), thickness=2) # Top row
   (L-R) - Face 2
```

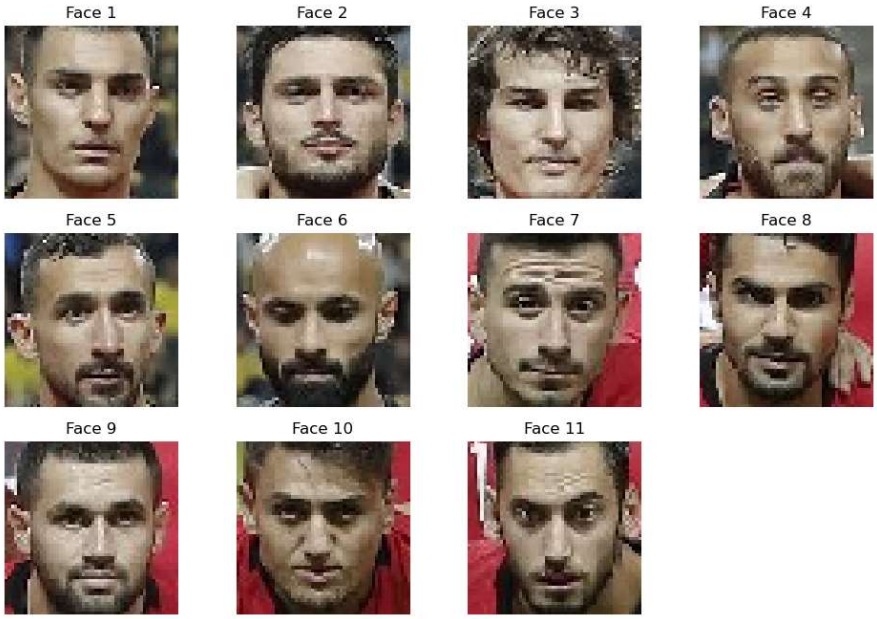
Your resulting image will look like this:

Step4: Save Cropped areas to a separate file and use imwrite in the cv2 library to write the files to the faces folder. A sample of the coding routine to do this is as follows:

```
# Initialize DataFrame to store cropped faces
cropped_faces = []

# Extract regions defined by circles and save them as separate images
for i, circle_param in enumerate(circle_parameters):
    center = circle_param["center"]
    radius = circle_param["radius"]
    x, y = center
    top_left = (x - radius, y - radius)
    bottom_right = (x + radius, y + radius)
    cropped_face = image[top_left[1]:bottom_right[1], top_left[0]:bottom_right[0]]
    cv2.imwrite(f"Dataset/faces/orgface_{i+1}.png", cropped_face)
    cropped_faces.append(cropped_face)
```

Your resulting files will look like this:



Step 5: Repeat steps 3 and 4 for all of the 5 images

Step 6: Build the model –

- Experiment with cascade classifier in the OS library using the haarcascade files that were loaded in Step 1.
- The script to do this is as follows:

```
import cv2
import os
import matplotlib.pyplot as plt

# Load built-in cascade classifier.
face_classifier_path = 'Dataset/haarcascade_frontalface_alt.xml'

# Load the cascade classifier
face_classifier = cv2.CascadeClassifier(face_classifier_path)

# Load the image
image_path = 'Dataset/sportsteam.png'

# Read the image
img = cv2.imread(image_path)

# Convert the image to grayscale for face detection
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Detect faces
faces = face_classifier.detectMultiScale(gray, 1.05, 3)
```

README

```
# Print the number of faces detected
print("Number of faces detected: ", len(faces))

# Draw rectangles around the detected faces
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
for (x, y, w, h) in faces:
    cv2.rectangle(img, (x, y), (x+w, y+h), (127, 0, 255), 2)

# Display the image with rectangles around faces
plt.imshow(img)
```

- Resize all Images in the Faces folder to ensure compliance with NumPy array requirements
- Load dataset and convert to numpy array
- Split Dataframes into Test and Training files
- Train the Model- Compile the model with DNN

```
# Convert faces and labels lists to NumPy arrays
faces = np.array(faces)
labels = np.array(labels)

# Split the dataset into training and testing sets
train_faces, test_faces, train_labels, test_labels = train_test_split(faces, labels, test_size=0.2, random_state=42)

# Define the DNN model architecture
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(train_faces.shape[1], train_faces.shape[2], 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(np.max(labels) + 1, activation='softmax')  # Number of classes based on the maximum label
])

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
```

```
                metrics=['accuracy'])

# Normalize pixel values
train_faces = train_faces.astype('float32') / 255.0
test_faces = test_faces.astype('float32') / 255.0

# Reshape faces for input to CNN (add channel dimension)
train_faces = np.expand_dims(train_faces, axis=-1)
test_faces = np.expand_dims(test_faces, axis=-1)

# Train the model
model.fit(train_faces, train_labels, epochs=10, validation_data=(test_faces,
test_labels))
```

- Use the trained model for face recognition in the pedestrian_video.avi file

```
# Use the trained model for face recognition in videos
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
cap = cv2.VideoCapture('Dataset/pedestrian_video.avi')

```

- Open the video and place squares over where a face is recognized based on the model

```

    # Convert frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Detect faces using Haar cascade classifier
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5,
minSize=(30, 30))

    # Process each detected face
    for (x, y, w, h) in faces:
        # Extract the face region
        face = gray[y:y+h, x:x+w]

        # Resize and normalize the face image
        face = cv2.resize(face, (train_faces.shape[1], train_faces.shape[2]))
        face = face.astype('float32') / 255.0
        face = np.expand_dims(face, axis=-1)

        # Perform face recognition using the trained model
        predicted_label = np.argmax(model.predict(np.array([face])))

        # Draw bounding box and label on the frame
```

```
            cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
            cv2.putText(frame, str(predicted_label), (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0), 2)

        # Display the frame with bounding boxes and labels
        cv2.imshow('Face Recognition', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
```

ISSUES LOG:

- Kernal crashed on several occasions during the training of the model
- Make sure to check file locations and folder verification
- Only a few faces were identified by the model. I attribute it to having only 30+ images to build my model. An extensive image database with male, female, different cultures, shapes, and sizes is required to make this truly effective