

COMP6200 Machine Learning

Homework 9

Subash Gupta

April 3, 2024

ME-Civil (Graduate Student)

Contents

1	Introduction	1
2	Implementation	3
3	Experimental Results	5
4	Conclusion	11
A	Code	A.1

List of Figures

3.1	Original Image	5
3.2	Original Sample	5
3.3	Detected Face	6
3.4	Detected Face Sample	7
3.5	Edge Detected	8
3.6	Edge Detected Sample	9
3.7	Edge Detected	10
3.8	Edge Detected	10
A.1	Face Detection	A.1
A.2	Edge Detection	A.2
A.3	PCA Applying	A.3

A.4 Classification A.4

CHAPTER 1

Introduction

Face detection

Face detection using OpenCV is a widely adopted method in computer vision, leveraging the library's powerful features to identify and locate human faces within images or video streams. OpenCV, short for Open Source Computer Vision Library, provides a multitude of pre-trained classifiers for face detection, among which the Haar feature-based cascade classifiers are the most popular. These classifiers work by scanning the image at multiple scales, searching for face-like features. When a face is detected, the algorithm returns the coordinates of the rectangular region enclosing the face.

The process begins by converting the image to grayscale, as the Haar classifier requires grayscale images to operate. This conversion simplifies the image, reducing the computational complexity. Next, the Haar cascade function is called with parameters that dictate the scale of image reduction and the number of neighbors each candidate rectangle should have to retain it. This step is crucial for reducing false positives. When the function executes, it detects faces at different sizes, thanks to the scale parameter, making the detection more robust across various face sizes.

The output is a list of rectangles where faces were detected. Developers can then use this information for further processing, like recognizing specific facial features, applying masks, or identifying individuals, depending on the application's needs. OpenCV's face detection is highly efficient and can be run in real-time, making it suitable for applications ranging from security systems to user interface interactions in real-time environments.

Multiclass Classification

Multiclass classification using the Support Vector Machine (SVC) module in scikit-learn is a sophisticated approach for categorizing instances into one of several classes. SVC, originally designed for binary classification, is extended to handle multiple classes by employing strategies like one-vs-one or one-vs-all. In the one-vs-one approach, for a problem with N classes, $N \times (N - 1)/2$ classifiers are created, and each one is trained to distinguish between a pair of classes. On the other hand, the one-vs-all strategy trains a single classifier per class, with the classifier trained to distinguish the samples of one class from all other classes.

When using SVC for multiclass classification, you first need to choose a kernel type, such as linear, polynomial, or radial basis function (RBF), depending on the nature of your data and the relationship between features and class labels. The choice of kernel plays a crucial role in the effectiveness of the SVC model. After selecting the kernel, you train the model on a labeled dataset, allowing the SVC to learn the decision boundaries between the classes.

Once the model is trained, it can predict the class of new, unseen instances. The decision function used by the SVC evaluates which side of the decision boundary an instance falls on, thereby classifying it into one of the classes. This method is particularly useful in scenarios where the classes are not linearly separable, and the model needs to find complex patterns in the feature space to differentiate between classes effectively. The use of SVC in multiclass settings is prevalent in various domains, including image recognition, text classification, and biological data classification, demonstrating its versatility and robustness in handling complex classification tasks.

CHAPTER 2

Implementation

Environment Setup:

- Ensure Python is installed on the system.
- Install OpenCV using the package manager, pip: `pip install opencv-python`.

Haar Cascade Classifier:

- Obtain the pre-trained Haar Cascade XML file for face detection, typically named `haarcascade_frontalface_default.xml`. This file can be downloaded from the OpenCV GitHub repository or found within the OpenCV package.

Face Detection Procedure:

- **Step 1: Import Libraries**
 - Import necessary modules: `import cv2`.
- **Step 2: Load the Classifier**
 - Load the Haar Cascade XML file: `face_cascade = cv2.CascadeClassifier('haarcascade_')`
- **Step 3: Read the Image**
 - Read the image in which faces are to be detected: `img = cv2.imread('path_to_image')`.
- **Step 4: Convert to Grayscale**
 - Convert the image to grayscale as Haar Cascade works on grayscale images: `gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)`.
- **Step 5: Detect Faces**
 - Detect faces in the image: `faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5)`.
 - `scaleFactor` and `minNeighbors` are parameters to adjust the detection accuracy.
- **Step 6: Draw Rectangles Around Detected Faces**

- For each detected face, draw a rectangle around it:

```
for (x, y, w, h) in faces:  
    cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)
```

- **Step 7: Display the Result**

- Display the image with detected faces: `cv2.imshow('Face Detection', img)`.
- Wait for a key press to exit and close the window: `cv2.waitKey(0), cv2.destroyAllWindows()`.

CHAPTER 3

Experimental Results

Original Image

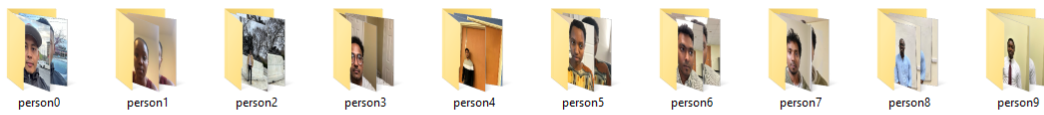


Fig. 3.1: Original Image



Fig. 3.2: Original Sample

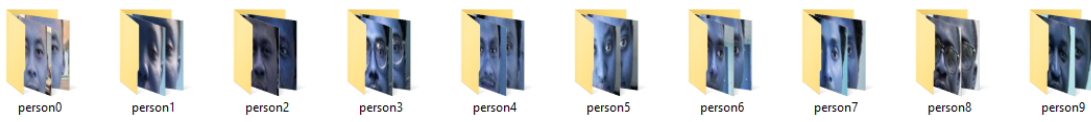


Fig. 3.3: Detected Face

Face Detected

Edge Detected

PCA

Multiclass Classification



Fig. 3.4: Detected Face Sample

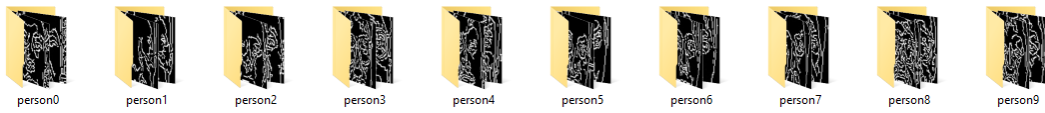


Fig. 3.5: Edge Detected



Fig. 3.6: Edge Detected Sample

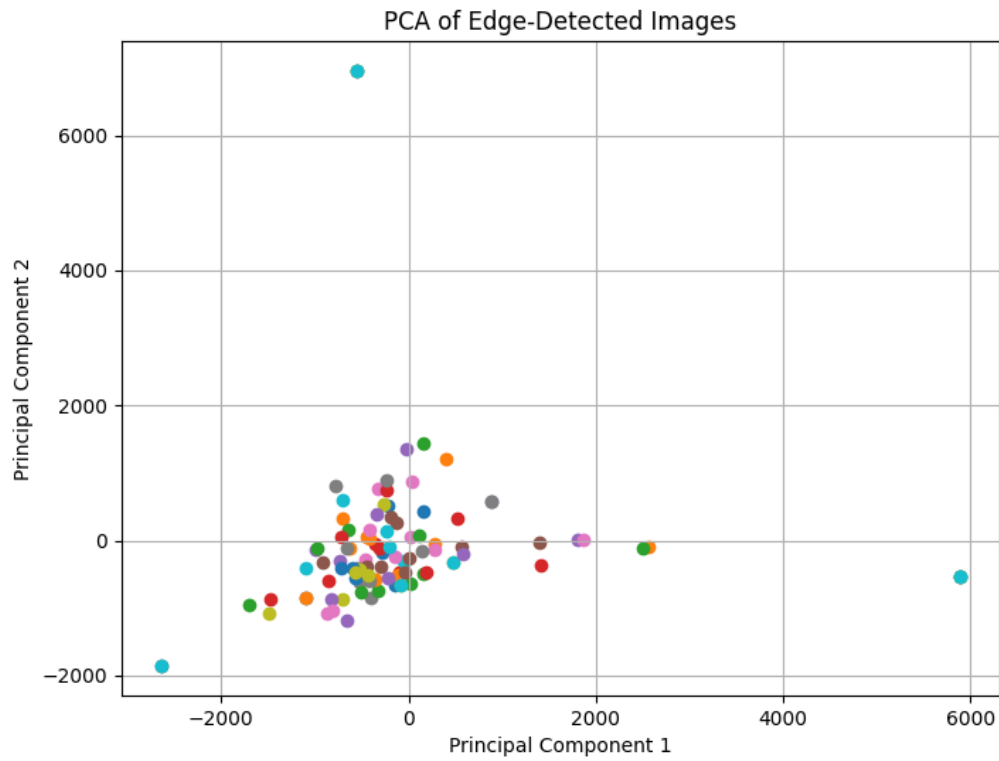


Fig. 3.7: Edge Detected

```
Python 3.11.5 (tags/v3.11.5:cce6ba9, Aug 24 2023, 14:38:34) [MSC v.1936 64 k
AMD64] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>
===== RESTART: C:\Users\sguptal\Desktop\assign\HW-9\5.py =====
[8 5 7 4 4 3 2 8 1 0 1 3 7 3 9 0 7 7 1 3]
>> |
```

Fig. 3.8: Edge Detected

CHAPTER 4

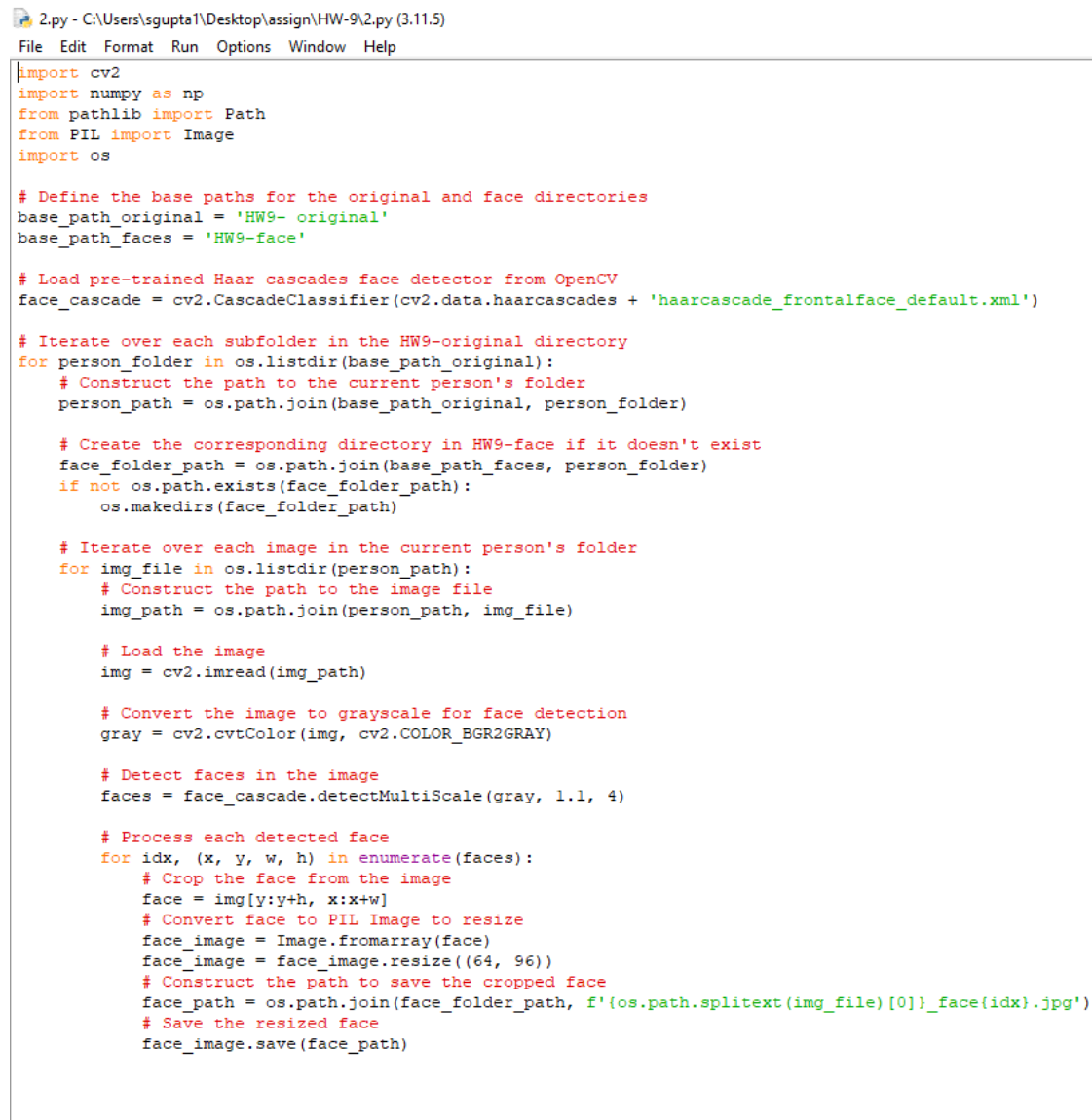
Conclusion

The faces were detected and edge detection was performed. The classification of image we performed after that.

APPENDIX A

Code

Question 2

A screenshot of a Python script titled '2.py' located at 'C:\Users\sgupta1\Desktop\assign\HW-9\2.py (3.11.5)'. The script is a face detection program using OpenCV. It imports cv2, numpy, pathlib, PIL Image, and os. It defines base paths for 'original' and 'face' directories. It loads a pre-trained Haar cascade face detector. It iterates over each subfolder in the 'original' directory, constructs the path to the current person's folder, and creates a corresponding folder in the 'face' directory if it doesn't exist. It then iterates over each image in the current person's folder, loads the image, converts it to grayscale, detects faces using the cascade classifier, and processes each detected face by cropping it, converting it to a PIL image, resizing it to (64, 96), and saving it in the 'face' directory with a filename like 'face_{idx}.jpg'.

```
2.py - C:\Users\sgupta1\Desktop\assign\HW-9\2.py (3.11.5)
File Edit Format Run Options Window Help

import cv2
import numpy as np
from pathlib import Path
from PIL import Image
import os

# Define the base paths for the original and face directories
base_path_original = 'HW9- original'
base_path_faces = 'HW9-face'

# Load pre-trained Haar cascades face detector from OpenCV
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

# Iterate over each subfolder in the HW9-original directory
for person_folder in os.listdir(base_path_original):
    # Construct the path to the current person's folder
    person_path = os.path.join(base_path_original, person_folder)

    # Create the corresponding directory in HW9-face if it doesn't exist
    face_folder_path = os.path.join(base_path_faces, person_folder)
    if not os.path.exists(face_folder_path):
        os.makedirs(face_folder_path)

    # Iterate over each image in the current person's folder
    for img_file in os.listdir(person_path):
        # Construct the path to the image file
        img_path = os.path.join(person_path, img_file)

        # Load the image
        img = cv2.imread(img_path)

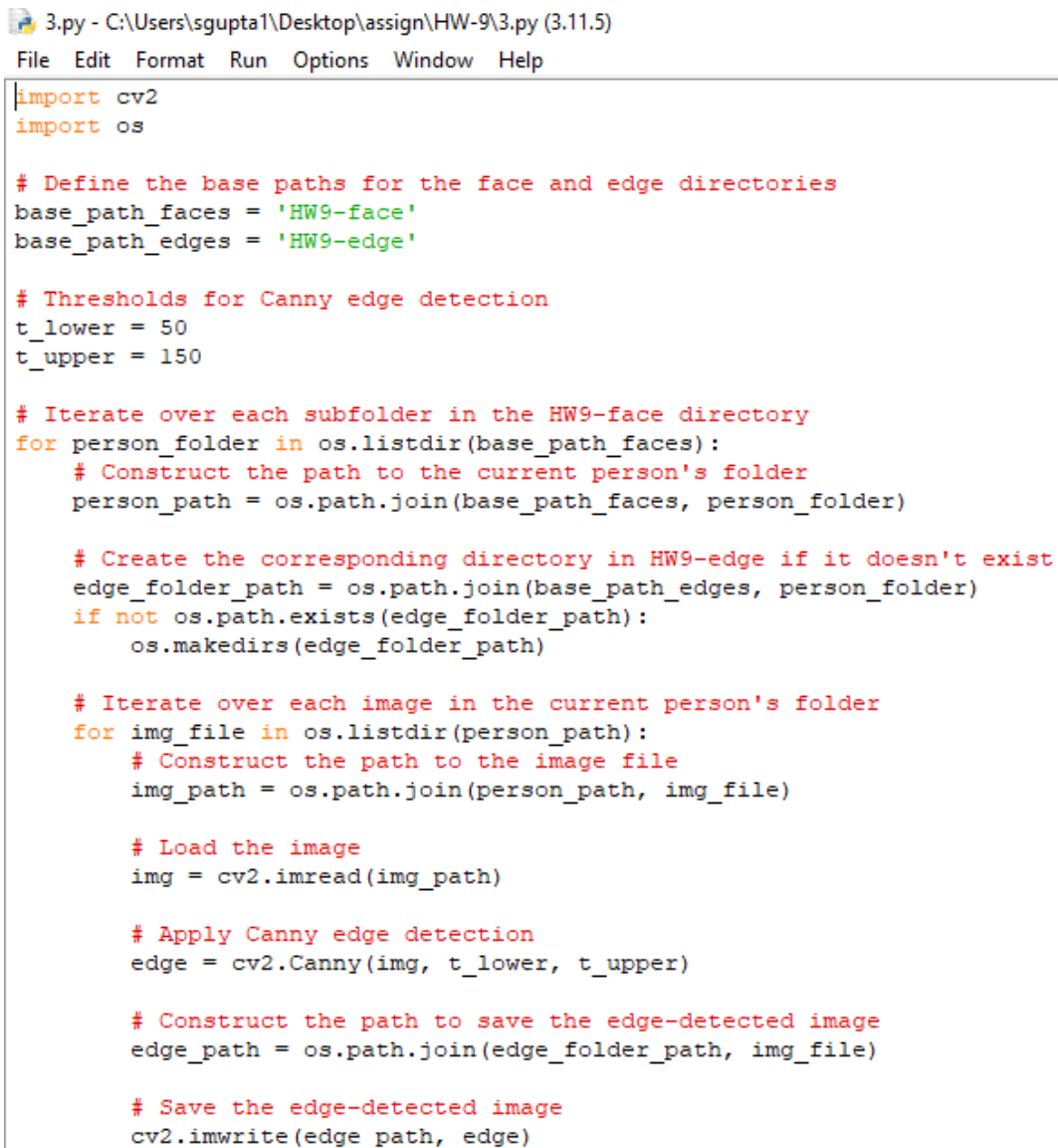
        # Convert the image to grayscale for face detection
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

        # Detect faces in the image
        faces = face_cascade.detectMultiScale(gray, 1.1, 4)

        # Process each detected face
        for idx, (x, y, w, h) in enumerate(faces):
            # Crop the face from the image
            face = img[y:y+h, x:x+w]
            # Convert face to PIL Image to resize
            face_image = Image.fromarray(face)
            face_image = face_image.resize((64, 96))
            # Construct the path to save the cropped face
            face_path = os.path.join(face_folder_path, f'{os.path.splitext(img_file)[0]}_face{idx}.jpg')
            # Save the resized face
            face_image.save(face_path)
```

Fig. A.1: Face Detection

Question 3

A screenshot of a Python script editor window titled '3.py - C:\Users\sgupta1\Desktop\assign\HW-9\3.py (3.11.5)'. The window has a menu bar with 'File', 'Edit', 'Format', 'Run', 'Options', 'Window', and 'Help'. The script content is as follows:

```
import cv2
import os

# Define the base paths for the face and edge directories
base_path_faces = 'HW9-face'
base_path_edges = 'HW9-edge'

# Thresholds for Canny edge detection
t_lower = 50
t_upper = 150

# Iterate over each subfolder in the HW9-face directory
for person_folder in os.listdir(base_path_faces):
    # Construct the path to the current person's folder
    person_path = os.path.join(base_path_faces, person_folder)

    # Create the corresponding directory in HW9-edge if it doesn't exist
    edge_folder_path = os.path.join(base_path_edges, person_folder)
    if not os.path.exists(edge_folder_path):
        os.makedirs(edge_folder_path)

    # Iterate over each image in the current person's folder
    for img_file in os.listdir(person_path):
        # Construct the path to the image file
        img_path = os.path.join(person_path, img_file)

        # Load the image
        img = cv2.imread(img_path)

        # Apply Canny edge detection
        edge = cv2.Canny(img, t_lower, t_upper)

        # Construct the path to save the edge-detected image
        edge_path = os.path.join(edge_folder_path, img_file)

        # Save the edge-detected image
        cv2.imwrite(edge_path, edge)
```

Fig. A.2: Edge Detection

Question 4

```

4.py - C:\Users\sgupta1\Desktop\assign\HW-9\4.py (3.11.5)
File Edit Format Run Options Window Help

import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from sklearn.decomposition import PCA
import os

# Base directory for the edge-detected images
base_dir = 'HW9-edge'

# List to store file paths
filepaths = []

# Iterate over each subfolder and collect image file paths
for person_folder in os.listdir(base_dir):
    person_path = os.path.join(base_dir, person_folder)
    for img_file in os.listdir(person_path):
        filepaths.append(os.path.join(person_path, img_file))

# Load images, convert to grayscale if necessary, and flatten them
images = []
for filepath in filepaths:
    img = Image.open(filepath)
    # Convert to grayscale if the image is not already in grayscale
    if img.mode != 'L':
        img = img.convert('L')
    images.append(np.array(img).flatten())

# Perform PCA with 200 components
pca = PCA(n_components=100)
principalComponents = pca.fit_transform(images)

# Plot the principal components
plt.figure(figsize=(8, 6))
for i, _ in enumerate(principalComponents):
    plt.scatter(principalComponents[i, 0], principalComponents[i, 1], marker='o')
plt.title('PCA of Edge-Detected Images')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.grid(True)
plt.show()

```

Fig. A.3: PCA Applying

Question 5

```

5.py - C:\Users\sgupta1\Desktop\assign\HW-9\5.py (3.11.5)
File Edit Format Run Options Window Help
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
import numpy as np
from PIL import Image
import os

# Initialize variables
images = []
labels = []

# Base directory for the processed images
base_dir = 'HW9-edge'

# Iterate over each subfolder and collect image file paths
for i, person_folder in enumerate(sorted(os.listdir(base_dir))):
    person_path = os.path.join(base_dir, person_folder)
    for img_file in os.listdir(person_path):
        # Construct the full file path
        filepath = os.path.join(person_path, img_file)
        # Load the image, convert to grayscale if necessary, and flatten it
        img = Image.open(filepath)
        if img.mode != 'L':
            img = img.convert('L')
        images.append(np.array(img).flatten())
        # Assign labels based on the person_folder name
        labels.append(i)

# Convert the lists to numpy arrays for processing with sklearn
X = np.array(images)
y = np.array(labels)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the SVC model
svc = SVC(kernel='linear') # You can experiment with different kernels

# Train the model
svc.fit(X_train, y_train)

# Predict on the test set
y_pred = svc.predict(X_test)

print(y_pred)

```

Fig. A.4: Classification