Project Python 101

FaceStream Verification System

M. Eng. Tran, Minh Hieu

Fall 2024

Contents

I. Demonstration

II. Programming

1. Installation Requirements
2. Code 1: Live "Face Detection" Stream
3. Code 2: Live FaceStream Verifier
4. Code 3: FaceStream Verification System

III. Extra requirements

I. Complete Code (40%)

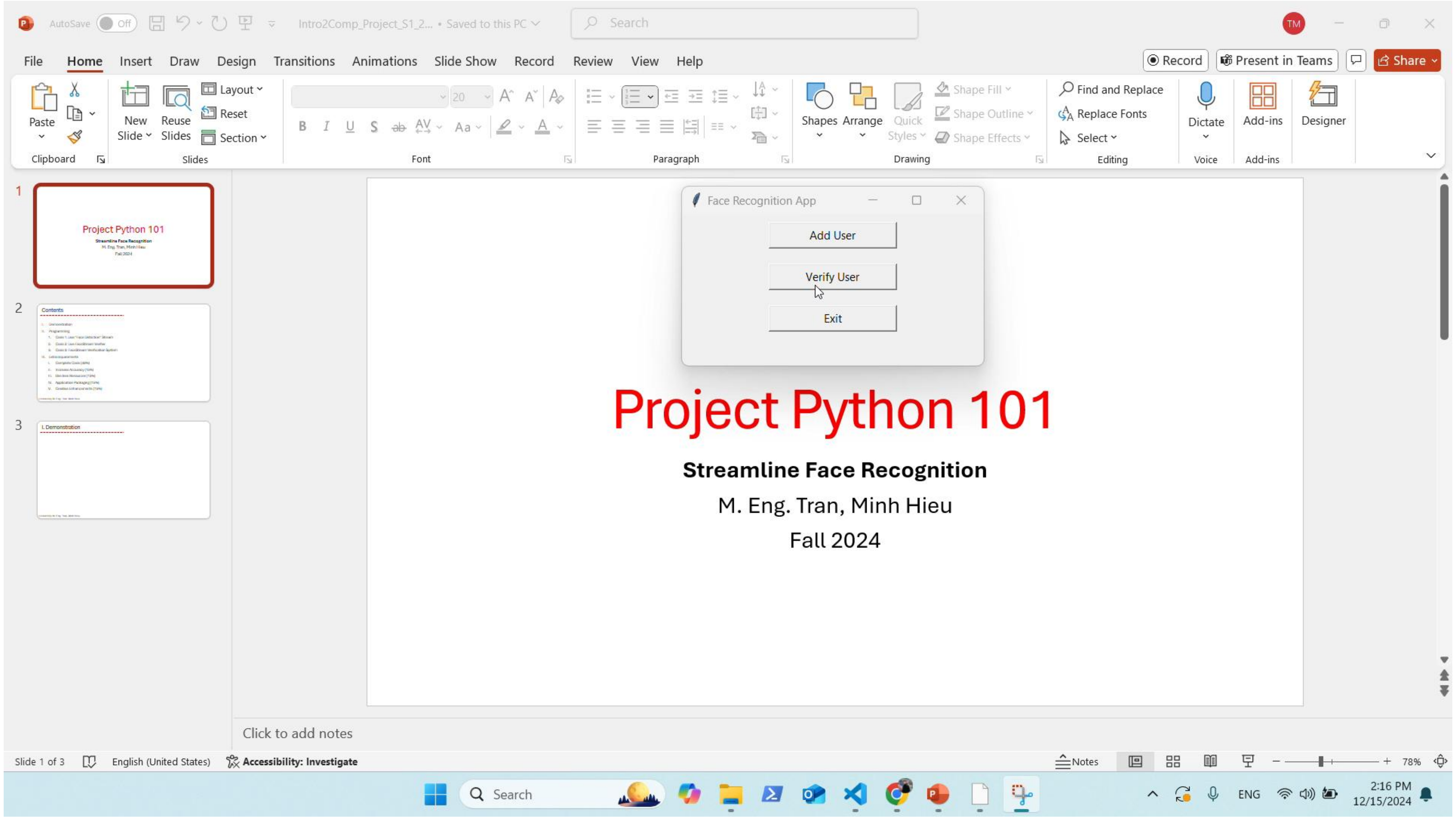
II. Increase Accuracy (15%)

III. Use less Resources (15%)

IV. Application Packaging (15%)

V. Creative Enhancements (15%)

I. Demonstration



Project Python 101

Streamline Face Recognition

M. Eng. Tran, Minh Hieu

Fall 2024

II. Programming

<u>Installation Requirements:</u>

1. Install:
   a) **OpenCV (cv2):** for computer vision tasks

      Command: **“pip install opencv-python”**

   b) **NumPy:** for numerical computations

      Command: **“pip install numpy”**

2. Built-In Libraries (No installation needed):
   a) **os**: a library to interact with the operating system

   b) **tkinter**: a library for creating graphical user interfaces (GUIs) in Python

   c) **threading**: a library to run multiple threads for parallel task execution

   d) **Time**: a library for handling time-related tasks

II. Programming

Code 1: Live "Face Detection" Stream (19 lines)

Demonstration

II. Programming

Code 1: Live "Face Detection" Stream:

Step 1: Load pre-built model and open webcam:

```
import cv2
import time
# Load the pre-trained Haar Cascade classifier for face detection
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
# Initialize the webcam (0 is the default camera)
cap = cv2.VideoCapture(0)
```

II. Programming

Code 1: Live "Face Detection" Stream:

Step 2: Read frames from webcam and detect faces:

```
while True:
    # Capture frame-by-frame
    ret, frame = cap.read()
    start_time = time.time()
    # Convert the frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Detect faces in the grayscale image
    faces = face_cascade.detectMultiScale(gray, 1.1, 4)
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
```

II. Programming

Code 1: Live "Face Detection" Stream:

Step 2: Read frames from webcam and detect faces:

```
while True:
    # Capture frame-by-frame
    ret, frame = cap.read()
    start_time = time.time()
    # Convert the frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Detect faces in the grayscale image
    faces = face_cascade.detectMultiScale(gray, 1.1, 4)
    for (x, y, w, h) in faces:
            cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
```

II. Programming

Code 1: Live "Face Detection" Stream:

Step 2: Read frames from webcam and detect faces:

```
while True:
    # Capture frame-by-frame
    ret, frame = cap.read()
    start_time = time.time()
    # Convert the frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Detect faces in the grayscale image
    faces = face_cascade.detectMultiScale(gray, 1.1, 4)
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
```

Read frames from webcam

Start time (for FPS calculation)

II. Programming

Code 1: Live "Face Detection" Stream:

Step 2: Read frames from webcam and detect faces:

```
while True:
    # Capture frame-by-frame
    ret, frame = cap.read()
    start_time = time.time()
    # Convert the frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Detect faces in the grayscale image
    faces = face_cascade.detectMultiScale(gray, 1.1, 4)
    for (x, y, w, h) in faces:
            cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
```

II. Programming

Code 1: Live "Face Detection" Stream:

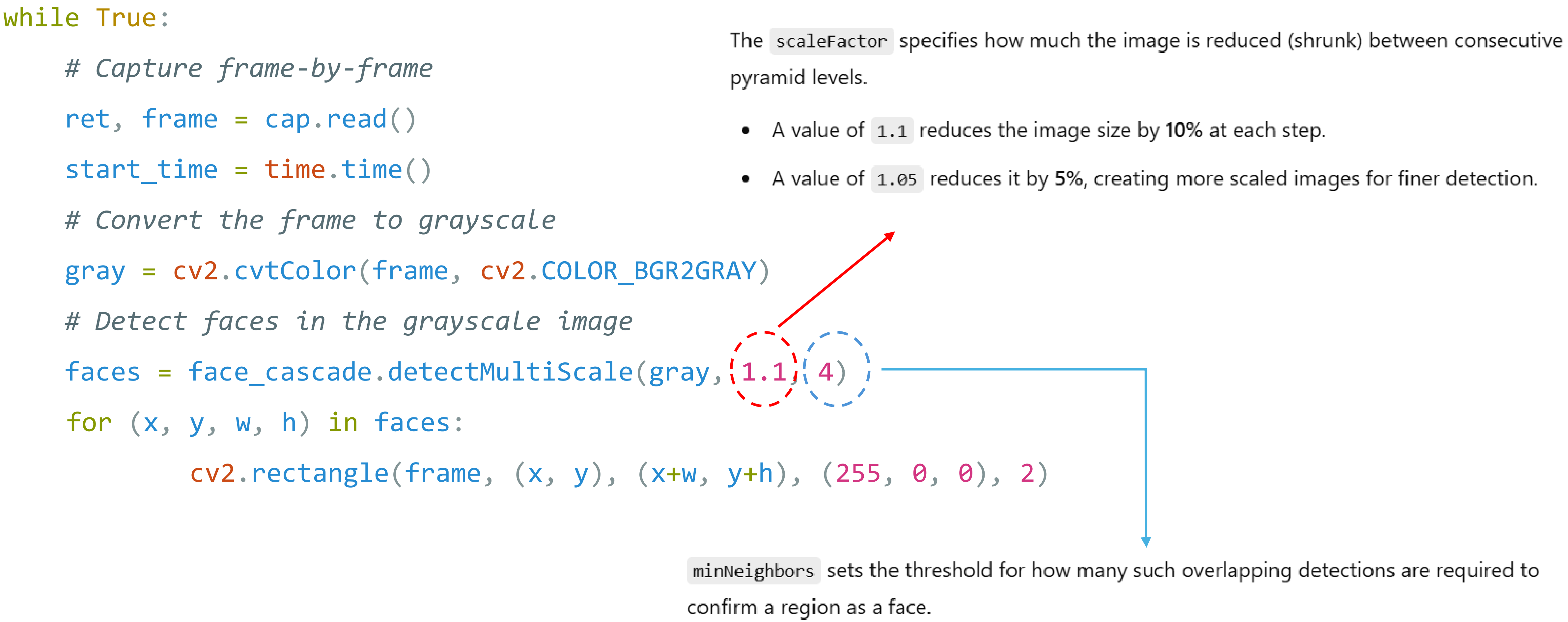
Step 2: Read frames from webcam and detect faces:

```
while True:
    # Capture frame-by-frame
    ret, frame = cap.read()
    start_time = time.time()
    # Convert the frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Detect faces in the grayscale image
    faces = face_cascade.detectMultiScale(gray, 1.1, 4)
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
```

II. Programming

Code 1: Live "Face Detection" Stream:

Step 2: Read frames from webcam and detect faces:



```python
while True:
    # Capture frame-by-frame
    ret, frame = cap.read()
    start_time = time.time()
    # Convert the frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Detect faces in the grayscale image
    faces = face_cascade.detectMultiScale(gray, 1.1, 4)
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
```

The `scaleFactor` specifies how much the image is reduced (shrunk) between consecutive pyramid levels.

- A value of `1.1` reduces the image size by **10%** at each step.
- A value of `1.05` reduces it by **5%**, creating more scaled images for finer detection.

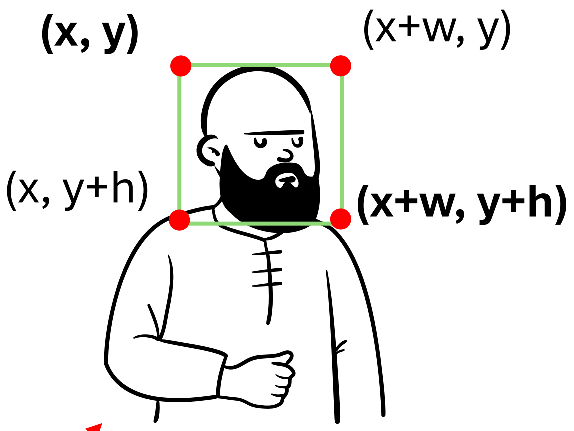
`minNeighbors` sets the threshold for how many such overlapping detections are required to confirm a region as a face.

II. Programming

Code 1: Live "Face Detection" Stream:

Step 2: Read frames from webcam and detect faces:

```
while True:
    # Capture frame-by-frame
    ret, frame = cap.read()
    start_time = time.time()
    # Convert the frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Detect faces in the grayscale image
    faces = face_cascade.detectMultiScale(gray, 1.1, 4)
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
```



Color of the box

thickness

II. Programming

Code 1: Live "Face Detection" Stream:

Step 3: Read frames from webcam and detect faces:

```
while True:
    ...
    end_time = time.time()
    fps = 1 / (end_time - start_time)
    cv2.putText(frame, f"FPS: {fps:.2f}", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    cv2.imshow('Face Recognition', frame)
    # Break the loop when the user presses 'q'
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# Release the capture and close the window
cap.release()
cv2.destroyAllWindows()
```

II. Programming

Code 1: Live "Face Detection" Stream:

Step 3: Read frames from webcam and detect faces:

```
while True:
    ...
    end_time = time.time()
    fps = 1 / (end_time - start_time)
    cv2.putText(frame, f"FPS: {fps:.2f}", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    cv2.imshow('Face Recognition', frame)
    # Break the loop when the user presses 'q'
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# Release the capture and close the window
cap.release()
cv2.destroyAllWindows()
```

End time and Calculate Fram-per-second

II. Programming

Code 1: Live "Face Detection" Stream:

Step 3: Read frames from webcam and detect faces:

```
while True:
    ...
    end_time = time.time()
    fps = 1 / (end_time - start_time)
    cv2.putText(frame, f"FPS: {fps:.2f}", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    cv2.imshow('Face Recognition', frame)
    # Break the loop when the user presses 'q'
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# Release the capture and close the window
cap.release()
cv2.destroyAllWindows()
```

Put the text “FPS” to frame at location (10, 30), font, fontsize, color and thickness

II. Programming

Code 1: Live "Face Detection" Stream:

Step 3: Read frames from webcam and detect faces:

```
while True:
    ...
    end_time = time.time()
    fps = 1 / (end_time - start_time)
    cv2.putText(frame, f"FPS: {fps:.2f}", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    cv2.imshow('Face Recognition', frame)
    # Break the loop when the user presses 'q'
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# Release the capture and close the window
cap.release()
cv2.destroyAllWindows()
```

Show the screen

II. Programming

Code 1: Live "Face Detection" Stream:

Step 3: Read frames from webcam and detect faces:

```
while True:
    ...
    end_time = time.time()
    fps = 1 / (end_time - start_time)
    cv2.putText(frame, f"FPS: {fps:.2f}", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    cv2.imshow('Face Recognition', frame)
    # Break the loop when the user presses 'q'
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# Release the capture and close the window
cap.release()
cv2.destroyAllWindows()
```

Press ‘q’ to quit

II. Programming

Code 1: Live "Face Detection" Stream:

Step 3: Read frames from webcam and detect faces:

```
while True:
    ...
    end_time = time.time()
    fps = 1 / (end_time - start_time)
    cv2.putText(frame, f"FPS: {fps:.2f}", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    cv2.imshow('Face Recognition', frame)
    # Break the loop when the user presses 'q'
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# Release the capture and close the window
cap.release()
cv2.destroyAllWindows()
```

Release and close windows

II. Programming

Code 2: Live FaceStream Verifier (71 lines)

Demonstration

II. Programming

Code 2: Live FaceStream Verifier:

Step 1: Set-up model and create data storage:

```
import cv2

import os

import numpy as np

import time

# Initialize the Haar Cascade for face detection

face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_default.xml")
```

```
if not os.path.exists('dataset'):

    os.makedirs('dataset’)
```

Create a directory to store the images if it doesn't exist

II. Programming

Code 2: Live FaceStream Verifier:

Step 2: Create a function to capture user images and store them with the associated name:

A1

```python
def capture_face_image():
    name = input("Enter your name: ")
    cap = cv2.VideoCapture(0)
    img_count = 0
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        # Convert to grayscale for better detection
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        # Detect faces
        faces = face_cascade.detectMultiScale(gray, 1.3, 5)
        ...
```

B1

```python
        ...
        # Draw rectangles around faces and save them
        for (x, y, w, h) in faces:
            img_count += 1
            face_image = frame[y:y+h, x:x+w]
            # Save the image with the associated name
            img_filename = f'dataset/{name}_{img_count}.jpg'
            cv2.imwrite(img_filename, face_image)
        # Stop after capturing 20 images (or any number)
        if img_count >= 20:
            break
    cap.release()
    cv2.destroyAllWindows()
    print("Face images saved!")
```

II. Programming

Code 2: Live FaceStream Verifier:

Step 3: Create a function to load and compare the captured faces:

A2

```
def match_face():
    cap = cv2.VideoCapture(0)
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        start_time = time.time()
        # Convert to grayscale for detection
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        # Detect faces
        faces = face_cascade.detectMultiScale(gray, 1.3, 5)
        for (x, y, w, h) in faces:
            face_image = frame[y:y+h, x:x+w]
        ...
```

B2

```
        ...
        for (x, y, w, h) in faces:
            face_image = frame[y:y+h, x:x+w] # Crop the face region
            # Compare this face with saved faces (this is a simple feature match using Euclidean distance)
            best_match = None
            min_distance = float('inf')
            # Iterate through the saved face images
            for filename in os.listdir('dataset'):
                stored_face = cv2.imread(os.path.join('dataset', filename))
                stored_face_gray = cv2.cvtColor(stored_face, cv2.COLOR_BGR2GRAY)
```

II. Programming

Code 2: Live FaceStream Verifier:

Step 3: Create a function to load and compare the captured faces:

B2

```
        ...
        for (x, y, w, h) in faces:
            face_image = frame[y:y+h, x:x+w] # Crop the face region
            # Compare this face with saved faces (this is a simple feature match using Euclidean distance)
            best_match = None
            min_distance = float('inf')
            # Iterate through the saved face images
            for filename in os.listdir('dataset'):
                stored_face = cv2.imread(os.path.join('dataset', filename))
                stored_face_gray = cv2.cvtColor(stored_face, cv2.COLOR_BGR2GRAY)
```

C2

```
        ...

                # Ensure that the face is the same size
                resized_stored_face = cv2.resize(stored_face_gray, (w, h))

                # Compute the Euclidean distance (simplistic approach)
                dist = np.linalg.norm(resized_stored_face - gray[y:y+h, x:x+w])
                if dist < min_distance:
                    min_distance = dist
                    best_match = filename
```

II. Programming

Code 2: Live FaceStream Verifier:

Step 3: Create a function to load and compare the captured faces:

C2

```
        ...

                # Ensure that the face is the same size
                resized_stored_face = cv2.resize(stored_face_gray, (w, h))

                # Compute the Euclidean distance (simplistic approach)
                dist = np.linalg.norm(resized_stored_face - gray[y:y+h, x:x+w])
                if dist < min_distance:
                    min_distance = dist
                    best_match = filename
```

D2

```
        ...

            # If a match is found
            if best_match:
                user_name = best_match.split('_')[0]
                # Extract name from the filename
                print(f"Match Found: {user_name}")
                # Adjust position of the name
                text_x = x + 10
                text_y = y - 10
                cv2.putText(frame, f"{user_name}", (text_x, text_y), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
            cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)
```

II. Programming

Code 2: Live FaceStream Verifier:

Step 3: Create a function to load and compare the captured faces:

D2

```
        ...

            # If a match is found
            if best_match:
                user_name = best_match.split('_')[0]
                # Extract name from the filename
                print(f"Match Found: {user_name}")
                # Adjust position of the name
                text_x = x + 10
                text_y = y - 10
                cv2.putText(frame, f"{user_name}", (text_x,
text_y), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
            cv2.rectangle(frame, (x, y), (x + w, y + h),
(255, 0, 0), 2)
```

E2

```
        ...

        end_time = time.time()
        fps = 1 / (end_time - start_time)
        cv2.putText(frame, f"FPS: {fps:.2f}", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

        cv2.imshow('Face Matching', frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()
```

II. Programming

Code 2: Live FaceStream Verifier:

Step 4: Main Program:

```
# Main Program
while True:
    choice = input("Enter 1 to capture face or 2 to match
face, or q to quit: ")
    if choice == '1':
        capture_face_image()
    elif choice == '2':
        match_face()
    elif choice == 'q':
        break
```

II. Programming

Code 3: FaceStream Verification System (88 lines)

Demonstration

II. Programming

Code 3: FaceStream Verification System:

Step 1: Set-up model and create data storage:

```python
import cv2
import os
import numpy as np
import tkinter as tk
from tkinter import messagebox, simpledialog
from threading import Thread
import time
# Initialize the Haar Cascade for face detection
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_default.xml")
# Create a directory to store the images if it doesn't exist
if not os.path.exists('dataset'):
    os.makedirs('dataset')
```

II. Programming

Code 3: FaceStream Verification System:

Step 2: Create a GUI tabs and a function for capturing user images:

A3

```
class FaceRecognitionApp:
    def __init__(self, root):
        self.root = root
        self.root.title("FaceStream Verification System")
        self.root.geometry("400x200")

        # Capture face button
        self.capture_button = tk.Button(self.root, text="Add
User", width=20, command=self.capture_face_image)
        self.capture_button.pack(pady=10)

        # Match face button
        self.match_button = tk.Button(self.root, text="Verify
User", width=20, command=self.match_face)
        self.match_button.pack(pady=10)

        # Exit button
        self.exit_button = tk.Button(self.root, text="Exit",
width=20, command=self.root.quit)
        self.exit_button.pack(pady=10)
```

B3

```
        ...
    def capture_face_image(self):
        # Use after() to call the dialog in the main thread
        self.root.after(0, self._capture_face_image)

    def _capture_face_image(self):
        name = simpledialog.askstring("Input", "Enter your
name:", parent=self.root)
        if not name:
            messagebox.showerror("Error", "Name is
required!")
            return
        cap = cv2.VideoCapture(0)
        img_count = 0
        while True:
            ret, frame = cap.read()
            if not ret:
                break

            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            faces = face_cascade.detectMultiScale(gray,
1.3, 5)
```

II. Programming

Code 3: FaceStream Verification System:

Step 2: Create a GUI tabs and a function for capturing user images:

B3

```
        ...
    def capture_face_image(self):
        # Use after() to call the dialog in the main thread
        self.root.after(0, self._capture_face_image)

    def _capture_face_image(self):
        name = simpledialog.askstring("Input", "Enter your
name:", parent=self.root)
        if not name:
            messagebox.showerror("Error", "Name is
required!")
            return
        cap = cv2.VideoCapture(0)
        img_count = 0
        while True:
            ret, frame = cap.read()
            if not ret:
                break

            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            faces = face_cascade.detectMultiScale(gray,
1.3, 5)
```

C3

```
        ...

            # Draw rectangles around faces and save them
            for (x, y, w, h) in faces:
                img_count += 1
                face_image = frame[y:y+h, x:x+w]  # Crop
the face region

                # Save the image with the associated name
                img_filename =
f'dataset/{name}_{img_count}.jpg'
                cv2.imwrite(img_filename, face_image)

            # Stop after capturing 20 images
            if img_count >= 30:
                break

        cap.release()
        cv2.destroyAllWindows()
        messagebox.showinfo("Success", "Face images
saved!")
```

II. Programming

Code 3: FaceStream Verification System:

Step 3: Create a to load and compare the captured faces :

C3

```
    ...

      # Draw rectangles around faces and save them
      for (x, y, w, h) in faces:
          img_count += 1
          face_image = frame[y:y+h, x:x+w]  # Crop
the face region

          # Save the image with the associated name
          img_filename =
f'dataset/{name}_{img_count}.jpg'
          cv2.imwrite(img_filename, face_image)

      # Stop after capturing 20 images
      if img_count >= 30:
          break

    cap.release()
    cv2.destroyAllWindows()
    messagebox.showinfo("Success", "Face images
saved!")
```

D3

```
    ...

  def match_face(self):
      def match():
          cap = cv2.VideoCapture(0)
          while True:
              ret, frame = cap.read()
              if not ret:
                  break
              start_time = time.time()

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

              # Detect faces
              faces = face_cascade.detectMultiScale(gray,
1.3, 5)

              for (x, y, w, h) in faces:
                  face_image = frame[y:y+h, x:x+w]  #
Crop the face region
```

II. Programming

Code 3: FaceStream Verification System:

Step 3: Create a to load and compare the captured faces :

D3

```
...

    def match_face(self):
        def match():
            cap = cv2.VideoCapture(0)
            while True:
                ret, frame = cap.read()
                if not ret:
                    break
                start_time = time.time()

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

                # Detect faces
                faces = face_cascade.detectMultiScale(gray, 
1.3, 5)

                for (x, y, w, h) in faces:
                    face_image = frame[y:y+h, x:x+w]  # 
Crop the face region
```

E3

```
...

                    # Compare this face with saved faces
                    best_match = None
                    min_distance = float('inf')

                    # Iterate through the saved face images
                    for filename in os.listdir('dataset'):
                        stored_face = 
cv2.imread(os.path.join('dataset', filename))
                        stored_face_gray = 
cv2.cvtColor(stored_face, cv2.COLOR_BGR2GRAY)

                        # Ensure that face is the same size
                        resized_stored_face = 
cv2.resize(stored_face_gray, (w, h))

                        # Compute the Euclidean distance
                        dist = 
np.linalg.norm(resized_stored_face - gray[y:y+h, x:x+w])
                        if dist < min_distance:
                            min_distance = dist
                            best_match = filename
```

II. Programming

Code 3: FaceStream Verification System:

Step 3: Create a to load and compare the captured faces :

E3

```
        ...

                    # Compare this face with saved faces
                    best_match = None
                    min_distance = float('inf')

                    # Iterate through the saved face images
                    for filename in os.listdir('dataset'):
                        stored_face =
cv2.imread(os.path.join('dataset', filename))
                        stored_face_gray =
cv2.cvtColor(stored_face, cv2.COLOR_BGR2GRAY)

                        # Ensure that face is the same size
                        resized_stored_face =
cv2.resize(stored_face_gray, (w, h))

                        # Compute the Euclidean distance
                        dist =
np.linalg.norm(resized_stored_face - gray[y:y+h, x:x+w])
                        if dist < min_distance:
                            min_distance = dist
                            best_match = filename
```

F3

```
        ...

                    # If a match is found, display the name
                    if best_match:
                        user_name =
best_match.split('_')[0]  # Extract name from the filename
                        print(f"Match Found: {user_name}")
                        text_x = x + 10
                        text_y = y - 10
                        cv2.putText(frame, f"{user_name}",
(text_x, text_y), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255,
0), 2)

                    cv2.rectangle(frame, (x, y), (x + w, y
+ h), (255, 0, 0), 2)

                end_time = time.time()
                fps = 1 / (end_time - start_time)
                cv2.putText(frame, f"FPS: {fps:.2f}", (10,
30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

                cv2.imshow('Face Matching', frame)
```

II. Programming

Code 3: FaceStream Verification System:

Step 3: Create a to load and compare the captured faces :

F3

```
...

                    # If a match is found, display the name
                    if best_match:
                        user_name =
best_match.split('_')[0]  # Extract name from the filename
                        print(f"Match Found: {user_name}")
                        text_x = x + 10
                        text_y = y - 10
                        cv2.putText(frame, f"{user_name}",
(text_x, text_y), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255,
0), 2)

                    cv2.rectangle(frame, (x, y), (x + w, y
+ h), (255, 0, 0), 2)

                end_time = time.time()
                fps = 1 / (end_time - start_time)
                cv2.putText(frame, f"FPS: {fps:.2f}", (10,
30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

                cv2.imshow('Face Matching', frame)
```

G3

```
...

                if cv2.waitKey(1) & 0xFF == ord('q'):
                    break

            cap.release()
            cv2.destroyAllWindows()

        # Run match in a separate thread to prevent GUI
freezing
        match_thread = Thread(target=match)
        match_thread.start()

# Create the GUI application window
root = tk.Tk()
app = FaceRecognitionApp(root)

# Run the Tkinter event loop
root.mainloop()
```

III. Extra requirements

a) Complete Code (40%)

b) Increase Accuracy (15%)

c) Use less Resources (15%)

d) Application Packaging (15%)

e) Creative Enhancements (15%)

End of the Tutorial

Lectured by M. Eng. Tran, Minh Hieu