Facultad Regional Rosario
Universidad Tecnológica Nacional

Cátedra: Soporte

4º Año Ingeniería en Sistemas de Información

Presentación - OpenCV

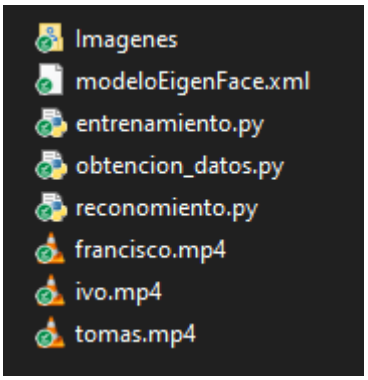
Instructivo Live Demo

Grupo Nº: 2
Integrantes:

- D'Ursi Ivo (46840)
- Mendiburu Francisco (47218)
- Tomás Peresin (46338)

Estructura del proyecto

Para la realización del Live Demo, se estructuró el proyecto de la siguiente manera:

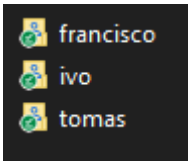


Videos

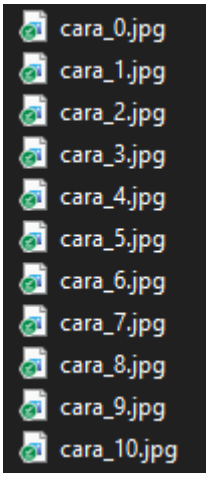
Constan de grabaciones de 15 segundos de los integrantes del grupo en frente de la cámara con el fin grabar sus rostros en distintos ángulos y así obtener las imágenes que luego se utilizarán para entrenar al modelo.

Imágenes

Carpeta donde se almacenan las imágenes de los rostros de cada integrante. Tiene la siguiente estructura:



En cada una de las carpetas se encuentran almacenadas 300 imágenes de 150x150p extraídas de los videos, cada una numerada:



modeloEigenFace.xml

Modelo de reconocimiento facial ya entrenado utilizado para crear una aplicación capaz de detectar a cada uno de los integrantes del grupo

obtencion_datos.py

Script utilizado para extraer las imágenes de los videos anteriormente mencionados las cuales son almacenadas en la carpeta imágenes

entrenamiento.py

Script utilizado para realizar el entrenamiento del modelo. Utiliza como entrada las imágenes de la carpeta imágenes y tiene como salida el modelo en un archivo.xml

reconomiento.py

Script en donde se emplea el modelo anteriormente entrenado para realizar un reconocimiento facial mediante la webcam de la PC.

Funcionamiento

obtencion_datos.py

Imports

```
import cv2
import imutils
import os
```

Código

Lo primero que realiza el script es asignar un reconocedor de rostros a la variable “clasificador_caras” y los distintos videos a la variable “cap”. El reconocedor de rostros ya viene incluido dentro de la librería de OpenCV y se encarga de simplemente detectar dónde hay rostros en una imagen.

```
clasificador_caras = cv2.CascadeClassifier(cv2.data.haarcascades+"haarcascade_frontalface_default.xml")
cap = cv2.VideoCapture("ivo.mp4")
```

Luego, se asegura de generar la ubicación en donde se guardarán las imágenes obtenidas del video utilizando la librería “os”.

```
nombre_persona = "ivo"
path_datos = "Imagenes"
path_persona = path_datos + "/" + nombre_persona

if not os.path.exists(path_persona):
    print("Carpeta Creada: ", path_persona)
    os.makedirs(path_persona)
```

Además de crear un contador inicializado en 0 e iniciar el loop utilizado para leer los fotogramas del video.

Una vez dentro del loop, se toman los frames del video y se acomodan con la librería “imutils” para que cada fotograma de cada video sea similar. (En nuestro caso tuvimos que bajar la resolución y rotar la imagen). Además, se copia un fotograma auxiliar y el original se pasa a escala de grises para mejorar la detección de rostros.

```
ref, frame = cap.read()
frame_achicado = imutils.resize(frame, height=640)
frame_rotado_achicado = imutils.rotate(frame_achicado, 0)

frame_auxiliar = frame_rotado_achicado.copy()

gris = cv2.cvtColor(frame_rotado_achicado, cv2.COLOR_BGR2GRAY)
```

Luego se invoca al método “detectMultiScale” del objeto “clasificador_caras” el cual devuelve una lista con las coordenadas de cada cara detectada en el frame. Estas coordenadas son utilizadas para hacer un recorte en la imagen y guardarlo en la carpeta Imágenes (También se delimita el rostro en el fotograma para asegurarnos visualmente de las caras que el clasificador detecta).

```
caras = clasificador_caras.detectMultiScale(gris,
    scaleFactor = 1.1,
    minNeighbors = 5,
    minSize = (100,100))

for (x, y, w, h) in caras:
    cara = frame_auxiliar[y:y+h, x:x+w]
    cara = cv2.resize(cara, (150,150), interpolation=cv2.INTER_CUBIC)
    cv2.rectangle(frame_rotado_achicado, (x,y), (x + w, y + h), (0,255,0), 2)
    cv2.imwrite(f"Imagenes/{nombre_persona}/cara_{count}.jpg", cara)
    count += 1
```

Finalmente se muestra el fotograma y se definen las condiciones de terminación del script. En nuestro caso determinamos obtener 300 imágenes de cada video, llevando la cuenta con la variable “count”.

```
cv2.imshow(nombre_persona, frame_rotado_achicado)

tecla = cv2.waitKey(1)
if tecla == ord("q") or ref == False or count >= 300:
    break
```

entrenamiento.py

Imports

```
import cv2
import os
import numpy as np
```

Código

En este script primero se define la ruta de la cual obtener las imágenes y además , a partir de esta, una lista con cada uno de los nombres de los integrantes para realizar las etiquetas de cada imagen.

```
path_datos = "Imagenes"
lista_personas = [x for x in os.listdir(path_datos) if not os.path.isfile(x)]
print("Lista de personas: ", lista_personas)

labels = []
data_caras = []
label = 0
```

Además se inicializan las variables en donde se almacenan las etiquetas, la información de la cara y la etiqueta actual.

Luego, por cada una de las personas en la lista, se procede a leer cada una de las imágenes, cambiarlas a escala de grises (requerimiento del modelo utilizado “Eigenfaces”) y la información se guarda en la lista “data_caras”. También se guarda en la lista “labels” la etiqueta asignada a cada imagen. Se muestra la imagen para asegurarnos visualmente que las imágenes se leen correctamente.

```
for nombre in lista_personas:
    path_persona = path_datos + "/" + nombre
    print("Leyendo imagenes")

    lista_archivos = os.listdir(path_persona)
    lista_archivos.pop()

    for archivo in lista_archivos:
        print("Caras: ", nombre + "/" + archivo)
        labels.append(label)
        data_caras.append(cv2.imread(path_persona+"/"+archivo,0))
        imagen = cv2.imread(path_persona+"/"+archivo,0)
        cv2.imshow("imagen", imagen)
        cv2.waitKey(10)
    label += 1
```

(el método “pop” lo utilizamos para eliminar uno de los archivos que crea drive automáticamente, en situaciones normales no es necesario)

Finalmente, una vez realizadas las etiquetas, se procede a entrenar el modelo. Para esto utilizamos los métodos ya existentes en OpenCV para crear el modelo, cargarle los datos de entrada, entrenarlo y almacenarlo en un archivo .xml

```
reconocedor_caras = cv2.face.EigenFaceRecognizer_create()

print("Entrenando...")
reconocedor_caras.train(data_caras, np.array(labels))

reconocedor_caras.write("modeloEigenFace.xml")
print("Modelo almacenado")
```

reconocedor.py

Imports

```
import cv2
import os
```

Código

En este script primero se vuelve a generar la ruta de las imágenes para obtener los nombres de los integrantes, luego se almacena el reconocedor de rostros básico utilizado en el primer script y el reconocedor realizado con el modelo “Eigenfaces”. El primero se utiliza para detectar el rostro y el segundo para detectar a quién pertenece.

```
path_datos = "Imagenes"
lista_personas = [x for x in os.listdir(path_datos) if not os.path.isfile(x)]
print("Lista de personas: ", lista_personas)


clasificador_caras = cv2.CascadeClassifier(cv2.data.haarcascades+"haarcascade_frontalface_default.xml")
cap = cv2.VideoCapture(0)

reconocedor_caras = cv2.face.EigenFaceRecognizer_create()
reconocedor_caras.read("modeloEigenFace.xml")
```

Luego el funcionamiento es muy similar al script de obtención de datos. Lo importante a destacar es que la cara obtenida del fotograma es convertida a escala de grises (requerimiento del modelo) y luego se llama al método “predict” del modelo teniendo como entrada el rostro y devolviendo una tupla con la etiqueta predicta y el grado de confianza.

```
for (x, y, w, h) in caras:
    cara = frame[y:y+h, x:x+w]
    cara_gris = cv2.cvtColor(cara, cv2.COLOR_BGR2GRAY)
    cara_gris_reducida = cv2.resize(cara_gris, (150,150), interpolation=cv2.INTER_CUBIC)
    resultado = reconocedor_caras.predict(cara_gris_reducida)
```

A partir de esta salida, se determina el texto a mostrar en el frame. Primero se determina si el grado de confianza es suficiente para determinar una correcta predicción. Si lo es, se utiliza la lista de nombres obtenida al principio la cual se relaciona con cada uno de los labels para finalmente determinar a quién pertenece el rostro. Si no es suficiente se muestra un texto en el cual se informa que no es posible detectar a la persona.

```
if resultado[1] < 7000:
    cv2.putText(frame, f"{lista_personas[resultado[0]]}", (x,y-5), 1,1.3,(255,255,0),1,cv2.LINE_AA)
    cv2.rectangle(frame, (x,y), (x + w, y + h), (0,255,0), 2)
else:
    cv2.putText(frame, "No se reconoce", (x,y-5), 1,1.3,(255,255,0),1,cv2.LINE_AA)
    cv2.rectangle(frame, (x,y), (x + w, y + h), (0,0,255), 2)
```

(Esta condición se encuentra dentro del for de las caras)
Finalmente se muestra la predicción y se definen las condiciones de término del script.

```
cv2.imshow("Prediccion", frame)

tecla = cv2.waitKey(1)
if tecla == ord("q") or ref == False:
    break
```

Observaciones

Luego de explicado el proyecto es importante realizar algunas observaciones:

- En caso de ya tener las imágenes de los rostros a utilizar con anterioridad, no es necesario realizar el script “obtencion_datos”.
- De la misma manera, si los datos ya se encuentran debidamente organizados y las capturas son similares, no es necesario utilizar las librerías “os” y “imutils”
- Todos los métodos utilizados de la librería “imutils” se encuentran también en OpenCV, aunque son un poco más complicados de utilizar.
- El nivel de éxito del proyecto depende mucho de las imágenes utilizadas para entrenar al modelo y aquellas utilizadas para la predicción. Entre mayor similitud haya entre estas mayor será el grado de confianza. Por eso se recomienda utilizar la misma cámara y de ser posible la misma iluminación.

Código completo

```
import cv2
import imutils
import os

clasificador_caras =
cv2.CascadeClassifier(cv2.data.haarcascades+"haarcascade_frontalface_default.x
ml")
cap = cv2.VideoCapture("ivo.mp4")

nombre_persona = "ivo"
path_datos = "Imagenes"
path_persona = path_datos + "/" + nombre_persona

if not os.path.exists(path_persona):
    print("Carpeta Creada: ", path_persona)
    os.makedirs(path_persona)

count = 0

while True:

    ref, frame = cap.read()
    frame_achicado = imutils.resize(frame, height=640)
    frame_rotado_achicado = imutils.rotate(frame_achicado, 0)

    frame_auxiliar = frame_rotado_achicado.copy()

    gris = cv2.cvtColor(frame_rotado_achicado, cv2.COLOR_BGR2GRAY)


    caras = clasificador_caras.detectMultiScale(gris,
        scaleFactor = 1.1,
        minNeighbors = 5,
        minSize = (100,100))

    for (x, y, w, h) in caras:
        cara = frame_auxiliar[y:y+h, x:x+w]
        cara = cv2.resize(cara, (150,150), interpolation=cv2.INTER_CUBIC)
        cv2.rectangle(frame_rotado_achicado, (x,y), (x + w, y + h), (0,255,0),
2)
```

```
        cv2.imwrite(f"Imagenes/{nombre_persona}/cara_{count}.jpg", cara)
        count += 1

    cv2.imshow(nombre_persona, frame_rotado_achicado)

    tecla = cv2.waitKey(1)
    if tecla == ord("q") or ref == False or count >= 300:
        break


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

```
import cv2
import os
import numpy as np

path_datos = "Imagenes"
lista_personas = [x for x in os.listdir(path_datos) if not os.path.isfile(x)]
print("Lista de personas: ", lista_personas)

labels = []
data_caras = []
label = 0

for nombre in lista_personas:
    path_persona = path_datos + "/" + nombre
    print("Leyendo imagenes")

    lista_archivos = os.listdir(path_persona)
    lista_archivos.pop()

    for archivo in lista_archivos:
        print("Caras: ", nombre + "/" + archivo)
        labels.append(label)
        data_caras.append(cv2.imread(path_persona+"/"+archivo,0))
        imagen = cv2.imread(path_persona+"/"+archivo,0)
        cv2.imshow("imagen", imagen)
        cv2.waitKey(10)
    label += 1
```

```
#print("Labels: ", labels)

reconocedor_caras = cv2.face.EigenFaceRecognizer_create()

print("Entrenando...")
reconocedor_caras.train(data_caras, np.array(labels))

reconocedor_caras.write("modeloEigenFace.xml")
print("Modelo almacenado")
```

```
import cv2
import os


path_datos = "Imagenes"
lista_personas = [x for x in os.listdir(path_datos) if not os.path.isfile(x)]
print("Lista de personas: ", lista_personas)


clasificador_caras =
cv2.CascadeClassifier(cv2.data.haarcascades+"haarcascade_frontalface_default.x
ml")
cap = cv2.VideoCapture(0)

reconocedor_caras = cv2.face.EigenFaceRecognizer_create()
reconocedor_caras.read("modeloEigenFace.xml")

while True:
    ref, frame = cap.read()

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    caras = clasificador_caras.detectMultiScale(gray,
        scaleFactor = 1.1,
        minNeighbors = 5,
        minSize = (100,100))

    for (x, y, w, h) in caras:
        cara = frame[y:y+h, x:x+w]
        cara_gris = cv2.cvtColor(cara, cv2.COLOR_BGR2GRAY)
```

```
        cara_gris_reducida = cv2.resize(cara_gris, (150,150),
interpolation=cv2.INTER_CUBIC)
        resultado = reconocedor_caras.predict(cara_gris_reducida)

        if resultado[1] < 7000:
            cv2.putText(frame, f"{lista_personas[resultado[0]]}", (x,y-5),
1,1.3,(255,255,0),1,cv2.LINE_AA)
            cv2.rectangle(frame, (x,y), (x + w, y + h), (0,255,0), 2)
        else:
            cv2.putText(frame, "No se reconoce", (x,y-5),
1,1.3,(255,255,0),1,cv2.LINE_AA)
            cv2.rectangle(frame, (x,y), (x + w, y + h), (0,0,255), 2)

    cv2.imshow("Prediccion", frame)

    tecla = cv2.waitKey(1)
    if tecla == ord("q") or ref == False:
        break


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