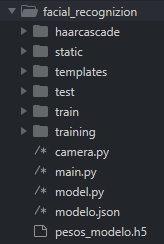
I) El sistema está estructurado de la siguiente manera:



1. haarcascade: Esta carpeta contiene al haarcascade, que es un archivo XML preentrenado para ayudar en la precisión del reconocimiento facial. Fue sacado del repositorio oficial de la librería OpenCV en GitHub. Referencia: <https://github.com/opencv/opencv/tree/master/data/haarcascades>
2. static: Esta carpeta contiene las imágenes usadas en la Web
3. templates: Esta carpeta contiene los archivos html, necesarios para mostrar el funcionamiento del sistema en la Web
4. test y train: Estas carpetas contienen subcarpetas de imágenes de las distintas expresiones para entrenar y validar en modelo. Estas imágenes fueron sacadas de Kaggle, dataset FER2013. Referencia: <https://www.kaggle.com/msambare/fer2013>
5. training: Esta carpeta contiene al archivo escrito en el lenguaje Python que se encargará de entrenar y validar el modelo.
6. camera.py: Este archivo nos ayudará a encender la cámara y detectar la cara en tiempo real
7. main.py: Este archivo es necesario para iniciar el llamado de la web y correr todo el sistema
8. model.py: Este archivo es el que predice en tiempo real las expresiones del rostro, haciendo uso de los archivos modelo.json, pesos\_modelo.h5 y el haarcascade
9. modelo.json: Este archivo es la representación escrita en formato JSON del modelo, una vez terminado
10. pesos\_modelo.h5: Este archivo va a contener los pesos resultantes del entrenamiento del modelo

II) Paquetes y librerías necesarios el sistema:

1. Pip: Herramienta utilizada para instalar librerías de Python de una forma más cómoda, más eficiente y más rápida.
2. Open CV: Librería libre de visión artificial originalmente desarrollada por INTEL. Se usa para integrar cámara de vídeo en aplicaciones escritas en Python
3. Tensorflow: Librería para compilar código de redes neuronales profundas. Si utilizamos la extensión GPU. Permitimos al ordenador que sea capaz de procesar a través de la unidad de procesamiento grafico de la propia tarjeta gráfica.
4. numpy: Librería de funciones matemáticas de alto nivel para operaciones con vectores y/o matrices.
5. seaborn: Es una librería para Python que permite generar fácilmente elegantes gráficos.
6. os: Módulo de Python para acceder a funcionalidades del Sistema operativo en el que corre la aplicación escrita en Python, por ejemplo, a funciones como leer y escribir ficheros.
7. tensorflow > paquete Keras: Librería de redes neuronales artificiales escrita en Python. Es la librería que contiene las diferentes técnicas que se harán uso en aprendizaje profundo.
8. livelossplot: Es una librería que permite la visualización de gráficas durante el entrenamiento de redes neuronales. Además de la visualización, mantiene los datos referentes a los valores mínimos y máximos obtenidos durante las ejecuciones.

III) Se instalarán los paquetes por medio de pip, por ejemplo, para instalar Tensorflow escribimos en la consola:

*pip install tensorflow*

IV) Se correrán los archivos del sistema, desde consola, navegando hasta la carpeta del proyecto y ejecutándolo con Phyon, por ejemplo, para ejecutar main.py:

*cd Users/users/Desktop/facial-recognizion*

*python main.py*

V) Documentación del código:

1. training/trainer.py: Archivo necesario para entrenar el modelo

# Importación de paquetes necesarios

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

import os

from tensorflow.keras.preprocessing.image import ImageDataGenerator, load\_img

from tensorflow.keras.layers import Dense, Input, Dropout,Flatten, Conv2D

from tensorflow.keras.layers import BatchNormalization, Activation, MaxPooling2D

from tensorflow.keras.models import Model, Sequential

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau

from livelossplot import PlotLossesKeras

from tensorflow.keras.utils import plot\_model

from IPython.display import SVG, Image

from livelossplot import PlotLossesKerasTF

from tensorflow.keras import backend as K

# Nos aseguramos de matar todas las sesiones de Keras existentes para empezar con una sesión nueva

K.clear\_session()

#Ploteando imágenes

# Aquí recorremos las imágenes de entrenamiento y evaluación y los redimensionamos a escala de 48x48 pixeles

def plot\_example\_images(plt):

    img\_size = 48

    plt.figure(0, figsize=(12,20))

    plt2.figure(0, figsize=(12,20))

    ctr = 0

    for expression in os.listdir("train/"):

        for i in range(1,6):

            ctr += 1

            plt.subplot(7,5,ctr)

            img = load\_img("train/" + expression + "/" +os.listdir("train/" + expression)[i], target\_size=(img\_size, img\_size))

            plt.imshow(img, cmap="gray")

    for expression in os.listdir("test/"):

        for i in range(1,6):

            ctr += 1

            plt2.subplot(7,5,ctr)

            img = load\_img("test/" + expression + "/" +os.listdir("test/" + expression)[i], target\_size=(img\_size, img\_size))

            plt2.imshow(img, cmap="gray")

    plt.tight\_layout()

    plt2.tight\_layout()

    return plt

# Imprimiendo carpetas de entrenamiento y validación

for expression in os.listdir("../train/"):

    print(str(len(os.listdir("../train/" + expression))) + " - Emoción de entrenamiento: " + expression)

for expression in os.listdir("../test/"):

    print(str(len(os.listdir("../test/" + expression))) + " - Emoción de validación: " + expression)

# Inicializamos el tamaño de las imágenes a 48X48 y la cantidad de lote de imágenes para procesar

img\_size = 48

batch\_size = 64

clases = 7 # Cantidad de carpetas de emociones, posibles predicciones

lr = 0.0005 # learning\_rate, ajustes de la red neuronal para acercarse a la predicción óptima

epochs = 50 # Cantidad de épocas que va a iterar el modelo

# Empezamos a preprocesar las imágenes

# Generando data para entrenamiento

datagen\_train = ImageDataGenerator(horizontal\_flip = True)

train\_generator = datagen\_train.flow\_from\_directory("../train/",

                                                    target\_size = (img\_size, img\_size),

                                                    color\_mode = "grayscale",

                                                    batch\_size = batch\_size,

                                                    class\_mode = "categorical", # Clasificación categórica por 7 emociones

                                                    shuffle = True)

# Generando data para evaluación

datagen\_validation = ImageDataGenerator(horizontal\_flip = True)

validation\_generator = datagen\_validation.flow\_from\_directory("../test/",

                                                    target\_size = (img\_size, img\_size),

                                                    color\_mode = "grayscale",

                                                    batch\_size = batch\_size,

                                                    class\_mode = "categorical", # Clasificación categórica por 7 emociones

                                                    shuffle = False)

# Creación del modelo de red neuronal convolucional

# Inicializamos en modelo

model = Sequential() # Secuencial porque va en varias capas apiladas, una depués de otra

# Agregamos las capas al modelo

model.add(Conv2D(64, (3, 3), padding = 'same', input\_shape = (img\_size, img\_size, 1))) #

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size = (2, 2))) #

model.add(Dropout(0.25))

model.add(Conv2D(128, (5, 5), padding = 'same'))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size = (2, 2)))

model.add(Dropout(0.25))

model.add(Conv2D(512, (3, 3), padding = 'same'))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size = (2, 2)))

model.add(Dropout(0.25))

model.add(Conv2D(512, (3, 3), padding = 'same'))

model.add(BatchNormalization())

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size = (2, 2)))

model.add(Dropout(0.25))

model.add(Flatten()) # Aquí aplanamos las imágenes muy profundas y pequeñas las aplanamos, una dimensión con toda la información

model.add(Dense(256))

model.add(BatchNormalization())

model.add(Activation('relu'))

# Bajamos el 25% de las neuronas para aprender caminos alternos, para adaptarse mejor a información nueva con diferentes emociones

model.add(Dropout(0.25))

model.add(Dense(512))

model.add(BatchNormalization())

model.add(Activation('relu'))

# Bajamos nuevamente el 25% de las neuronas

model.add(Dropout(0.25))

# Esta capa sirve para predecir entre las clases o emociones en el modelo. La probabilidad más alta es la predicción correcta

model.add(Dense(clases, activation = 'softmax'))

opt = Adam(lr = lr)

model.compile(optimizer = opt, loss = 'categorical\_crossentropy', metrics = ['accuracy'])

# Imprimimos las estadísticas del modelo

model.summary()

# Entrenando y evaluando el modelo

reduce\_lr = ReduceLROnPlateau(monitor = 'val\_loss', factor = 0.1, patience = 2, min\_lr = 0.00001, mode = 'auto')

# Guardamos los pesos obtenidos en cada época

checkpoint = ModelCheckpoint("../pesos\_modelo.h5", monitor = 'val\_accuracy', save\_weights\_only = True, mode = 'max', verbose = 1)

# Esto para imprimir la gráfica de pérdida y ganancia

callbacks = [PlotLossesKerasTF(), checkpoint, reduce\_lr]

history = model.fit(

    batch\_size = batch\_size,

    x = train\_generator,

    epochs = epochs,

    validation\_data = validation\_generator,

    callbacks = callbacks

)

# Representación de modelo entrenado en JSON, es la estructura del modelo, lo guardamos en model.json

model\_json = model.to\_json()

with open("../modelo.json", "w") as json\_file:

    json\_file.write(model\_json)

1. model.py: Archivo para predecir la emoción:

# Importamos paquetes necesarios

from tensorflow.keras.models import model\_from\_json

from tensorflow.python.keras.backend import set\_session

import numpy as np

import tensorflow as tf

# Configuración necesaria para optimizar la sesión de la cámara

config = tf.compat.v1.ConfigProto()

config.gpu\_options.per\_process\_gpu\_memory\_fraction = 0.15

session = tf.compat.v1.Session(config=config)

set\_session(session)

# Clase para predecir la emoción

class FacialExpressionModel(object):

    # Todas las posibles salidas, por orden de lectura de carpetas

    # LISTA\_EMOCIONES = ["Molesto", "Disgustado", "De miedo", "Feliz", "Neutral", "Triste", "Sorprendido"]

    # En este caso, solo usaremos la tristeza

    LISTA\_EMOCIONES = ["", "", "", "", "", "Triste", ""]

    # Inicializamos el modelo con los archivos obtenidos en el entrenamiento

    def \_\_init\_\_(self, model\_json\_file, model\_weights\_file):

        with open(model\_json\_file, "r") as json\_file:

            loaded\_model\_json = json\_file.read()

            self.loaded\_model = model\_from\_json(loaded\_model\_json)

        self.loaded\_model.load\_weights(model\_weights\_file)

    # Función para predecir

    def predict\_emotion(self, img):

        global session

        set\_session(session)

        # En este paso vamos a obtener una matriz con los porcentajes de las predicciones como [0, 0, 0, 0, 1, 0, 0], el 1 representa el valor más alto

        self.preds = self.loaded\_model.predict(img)

        # Los valores coinciden con el arreglo de emociones, seleccionamos la más alta y la retornamos, por ejemplo,  [0, 0, 0, 0, 1, 0, 0] => "Neutral"

        return FacialExpressionModel.LISTA\_EMOCIONES[np.argmax(self.preds)]

1. camera.py: Archivo para abrir la cámara del sistema

# Importamos paquetes necesarios

import cv2 # Librería OpenCV

from model import FacialExpressionModel # Improtamos el archivo model.py para predecir la emoción en la cara de la imagen

import numpy as np

# Cargamos el haarcascade

facec = cv2.CascadeClassifier('./haarcascade/haarcascade\_frontalface\_default.xml')

# Cargamos el archivo para predecir y le pasamos los archivos generados por el entrenamiento

model = FacialExpressionModel("modelo.json", "pesos\_modelo.h5")

# Cargamos el tipo de font de la cámara

font = cv2.FONT\_HERSHEY\_SIMPLEX

class VideoCamera(object):

    def \_\_init\_\_(self):

        self.video = cv2.VideoCapture(0) # Abrimos la cámara principal del sistema, si queremos otros, podemos probar con 1 y -1

    def \_\_del\_\_(self):

        self.video.release() # Terminar la sesión de cámara

    # returns camera frames along with bounding boxes and predictions

    def get\_frame(self):

        \_, fr = self.video.read() # Leemos la cámara

        gray\_fr = cv2.cvtColor(fr, cv2.COLOR\_BGR2GRAY) # Valor necesario para leer la cámara

        faces = facec.detectMultiScale(gray\_fr, 1.3, 5) # Para mejorar la detección de rostro

        # Algoritmo para abrir la cámara y retornarlo en archivo .jpeg

        for (x, y, w, h) in faces:

            fc = gray\_fr[y:y+h, x:x+w]

            roi = cv2.resize(fc, (48, 48))

            pred = model.predict\_emotion(roi[np.newaxis, :, :, np.newaxis])

            cv2.putText(fr, pred, (x, y), font, 1, (255, 255, 0), 2)

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

        \_, jpeg = cv2.imencode('.jpg', fr)

        return jpeg.tobytes()

1. main.py: Archivo para empezar a llamar al sistema

from flask import Flask, render\_template, Response # Flask para interpretar código python en la web mediante un puerto

from camera import VideoCamera # Invocamos a camera.py para abrir la cámara en la web

# Inicializamos Flask

app = Flask(\_\_name\_\_)

# Llamamos a la cámara del sistema

def gen(camera):

    while True:

        frame = camera.get\_frame()

        yield (b'--frame\r\n'

               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')

# Creamos las rutas para ser leídas por la web

@app.route('/') # Ruta principal

def index():

    return render\_template('index.html')

@app.route('/modelo') # Ruta para ver el modelo en un gráfico

def modelo():

    return render\_template('modelo.html')

@app.route('/video\_feed') # Ruta para llamar a la cámara

def video\_feed():

    return Response(gen(VideoCamera()), mimetype = 'multipart/x-mixed-replace; boundary=frame')

# Definimos puerto para desplegar sistema en la web y le definimos como en desarrollo True para detección de cambios

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(host='localhost', debug = True)