Proyecto final AI

Karla Ornelas Gamero

Cargamos librerias

```
import numpy as np
import os
import re
import matplotlib.pyplot as plt
#from google.colab import drive
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import keras
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
from keras.models import Sequential,Model
from tensorflow.keras.layers import Input
from keras.layers import Dense, Dropout, Flatten
#from keras.layers import Conv2D, MaxPooling2D
#from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (
    BatchNormalization, SeparableConv2D, MaxPooling2D, Activation, Flatten, Dropout, Dense, Conv2D
)
from keras.layers import LeakyReLU
```

Cargamos el dataset

Lo que hacemos en esta parte del código es donde cargamos las imágenes , en donde la ubicación de datasetcara busca fotos folder y va contando imagen por imagen que se ve en el folder. Busca si las extensión son jpg, jupeg, png,bmp,biff. Ademas de que cuenta las imágenes en cada directorio y almacena la información, también muestra el numero de directorios y de imágenes encontradas

```
import cv2

img = cv2.imread('/Users/karla/Documents/inteligencia_artificial/tareas/datasetcara')
#print(img.shape[0], img.shape[1], img.shape[2], len(img.shape))
dirname = os.path.join(os.getcwd(),'/Users/karla/Documents/inteligencia_artificial/tareas/datasetcara')
imgpath = dirname + os.sep

images = []
directories = []
dircount = []
prevRoot=''
cant=0

print("leyendo imagenes de ",imgpath)

for root, dirnames, filenames in os.walk(imgpath):
    for filename in filenames:
        if re.search("\.(jpg|jpeg|png|bmp|tiff)$", filename):
            cant=cant+1
            filepath = os.path.join(root, filename)
            image = plt.imread(filepath)
            if(len(image.shape)==3):

                images.append(image)
            b = "Leyendo..." + str(cant)
            print (b, end="\r")
            if prevRoot !=root:
                print(root, cant)
                prevRoot=root
                directories.append(root)
                dircount.append(cant)
                cant=0
dircount.append(cant)

dircount = dircount[1:]
dircount[0]=dircount[0]+1
```

```
print('Directorios leidos:',len(directories))
print("Imagenes en cada directorio", dircount)
print('suma Total de imagenes en subdirs:',sum(dircount))
```

```
leyendo imagenes de  c:/Users/karla/Documents/inteligencia_artificial/tareas/datasetcara\
c:/Users/karla/Documents/inteligencia_artificial/tareas/datasetcara\jessica 1
c:/Users/karla/Documents/inteligencia_artificial/tareas/datasetcara\jorge 6145
c:/Users/karla/Documents/inteligencia_artificial/tareas/datasetcara\karla 5413
c:/Users/karla/Documents/inteligencia_artificial/tareas/datasetcara\rodrigo 5114
c:/Users/karla/Documents/inteligencia_artificial/tareas/datasetcara\william 5278
Directorios leidos: 5
Imagenes en cada directorio [6146, 5413, 5114, 5278, 4691]
suma Total de imagenes en subdirs: 26642
```

Creamos las etiquetas

Esta parte del código es el que se encarga de crear la etiquetas de las imágenes en el cual las marca como labels, en el cual se recorre la lista con el dircount. Después se imprime la cantidad de directorios encontrados. Después creamos la lista de cara para guardar los nombres de los directorios encontrados

Después labels se convierte en un arreglo Numpy y y otro x con el tipo de datos uint8 con las imágenes, después se determinan la clases con la función unique(), y se imprime el numero total de salidas y la lista de clases encontradas

```
labels=[]
indice=0
for cantidad in dircount:
    for i in range(cantidad):
        labels.append(indice)
    indice=indice+1
print("Cantidad etiquetas creadas: ",len(labels))
caras=[]
indice=0
for directorio in directories:
    name = directorio.split(os.sep)
    print(indice , name[len(name)-1])
    caras.append(name[len(name)-1])
    indice=indice+1
y = np.array(labels)
X = np.array(images, dtype=np.uint8) #convierto de lista a numpy



# Find the unique numbers from the train labels
classes = np.unique(y)
nClasses = len(classes)
print('Total number of outputs : ', nClasses)
print('Output classes : ', classes)
```

```
Cantidad etiquetas creadas:  26642
```

```
0 jessica
1 jorge
2 karla
3 rodrigo
4 william
```

```
Total number of outputs :  5
Output classes :  [0 1 2 3 4]
```

Creamos Sets de Entrenamiento y Test

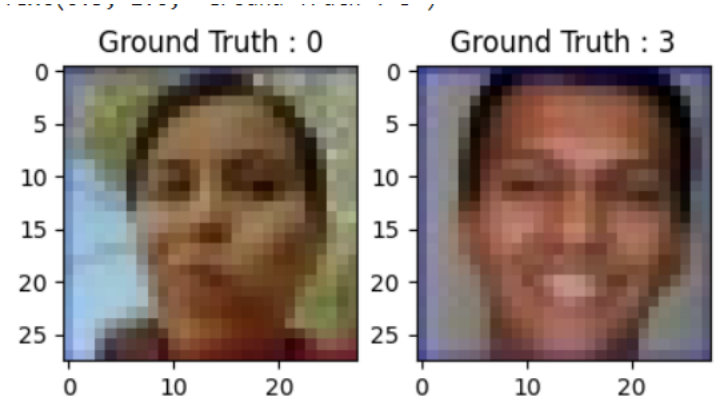
Se dividen los datos de prueba y de entrenamiento: el 80 % de entrenamiento y como se puede observar en el código de 0.2 significa que el 20% es de prueba

```
train_X,test_X,train_Y,test_Y = train_test_split(X,y,test_size=0.2)
print('Training data shape : ', train_X.shape, train_Y.shape)
print('Testing data shape : ', test_X.shape, test_Y.shape)
```

```
Training data shape :  (21313, 28, 28, 3) (21313,)
Testing data shape :  (5329, 28, 28, 3) (5329,)
```

En esta parte se muestra la primera imagen de los datos de entrenamiento y la primera imagen de los de prueba , la etiqueta ground truth significa de que clase es como jessy es 0 y rodrigo 3

```
plt.figure(figsize=[5,5])

# Display the first image in training data
plt.subplot(121)
plt.imshow(train_X[0,:,:], cmap='gray')
plt.title("Ground Truth : {}".format(train_Y[0]))

# Display the first image in testing data
plt.subplot(122)
plt.imshow(test_X[0,:,:], cmap='gray')
plt.title("Ground Truth : {}".format(test_Y[0]))
```

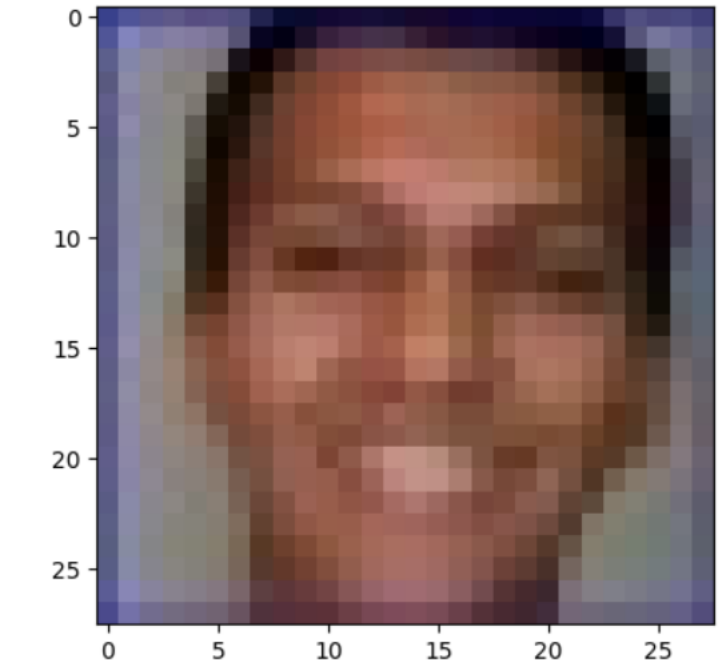


Preprocesamos las imágenes

Aquí se normaliza los datos de entrenamiento y prueba para que se pueda mostrar , primero se convierten en un float32 y después de dividen los pixel en 255 lo que hará que se normalicen y se convierten en un rango entre 0 y 1 para procesarlo que se puedan utilizar bien en el entrenamiento y aprendizaje

```
train_X = train_X.astype('float32')
test_X = test_X.astype('float32')
train_X = train_X/255.
test_X = test_X/255.
plt.imshow(test_X[0,:,:])
```

<matplotlib.image.AxesImage at 0x22f4d83c400>



Hacemos el One-hot Encoding para la red

En esta parte del código se realiza la representación one-hot para los datos , convirtiéndolas en vectores binarios, la conversión se realiza utiliza la funcion tp_categorical() , permitiendo que los modelos interpreten etiquetas de manera adecuada

```
train_Y_one_hot = to_categorical(train_Y)
test_Y_one_hot = to_categorical(test_Y)


print('Original label:', train_Y[0])
print('After conversion to one-hot:', train_Y_one_hot[0])
```

```
Original label: 0
After conversion to one-hot: [1. 0. 0. 0. 0.]
```

Creamos el Set de Entrenamiento y Validación

Esta parte del código crea grupos de entrenamiento y de la validación de los datos y las etiquetas, que evalúa el rendimiento del modelo en datos no vistos o no probados durante el entrenamiento, básicamente es para verificar la capacidad de la generalización del modelo

```
train_X,valid_X,train_label,valid_label = train_test_split(train_X, train_Y_one_hot, test_size=0.2, random_state=13)
print(train_X.shape,valid_X.shape,train_label.shape,valid_label.shape)
```

```
(17050, 28, 28, 3) (4263, 28, 28, 3) (17050, 5) (4263, 5)
```

Creamos el modelo de CNN

Por siguiente se configuran los parámetros y se creo un modelo de red neuronal convolucional para la clasificación La da el valor primero es la taza de aprendizaje, en numero de épocas de aprendizaje y el tamaño de imágenes que se utilizaran durante el entrenamiento. Tiene una estructura secuencial, las capas de clasificación se definen mediante capas densas con activación lineal y dropout. La última capa densa utiliza una activación softmax para clasificar las muestras en diferentes categorías.

Se muestra un resumen del modelo creado, que proporciona información sobre las capas y los parámetros. Finalmente, el modelo se compila utilizando el optimizador Stochastic Gradient Descent (SGD), con una función de pérdida categorical_crossentropy y se evalúa utilizando la métrica de precisión de accuracy

```
INIT_LR = 1e-3 # Valor inicial de learning rate. El valor 1e-3 corresponde con 0.001
epochs = 20 # Cantidad de iteraciones completas al conjunto de imagenes de entrenamiento
batch_size = 64 # cantidad de imágenes que se toman a la vez en memoria
sport_model = Sequential()
sport_model.add(Conv2D(32, kernel_size=(3, 3),activation='linear',padding='same',input_shape=(28,28,3)))
sport_model.add(LeakyReLU(alpha=0.1))
sport_model.add(MaxPooling2D((2, 2),padding='same'))
sport_model.add(Dropout(0.5))

sport_model.add(Flatten())
sport_model.add(Dense(32, activation='linear'))
sport_model.add(LeakyReLU(alpha=0.1))
sport_model.add(Dropout(0.5))
sport_model.add(Dense(nClasses, activation='softmax'))
sport_model.summary()
sport_model.compile(loss=keras.losses.categorical_crossentropy, optimizer=tf.keras.optimizers.legacy.SGD(learning_rate=INIT_LR, decay=INIT_
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        896

 leaky_re_lu (LeakyReLU)     (None, 28, 28, 32)        0

 max_pooling2d (MaxPooling2D  (None, 14, 14, 32)       0
 )

 dropout (Dropout)           (None, 14, 14, 32)        0

 flatten (Flatten)           (None, 6272)              0

 dense (Dense)               (None, 32)                200736

 leaky_re_lu_1 (LeakyReLU)   (None, 32)                0

 dropout_1 (Dropout)         (None, 32)                0

 dense_1 (Dense)             (None, 5)                 165

=================================================================
Total params: 201,797
Trainable params: 201,797
Non-trainable params: 0
_________________________________________________________________
```

Entrenamos el modelo: Aprende a clasificar imágenes

En esta parte del código se entrena la red convolucional utilizando los datos de entrenamiento y asi el modelo puede reconocer y clasificar las imágenes, el modelo fit() es el que lleva el entrenamiento a cabo, en el batch size se determina el lote de imágenes y la cantidad de épocas, verbosa muestra la información detallada del progreso del entrenamiento asi como la perdida y precisión de cada época. También se evalúa e rendimiento del modelo en los datos de validación durante el entrenamiento. después se almacenan los datos en la variable sport train. y posteriormente se guarda el modelo como un archivo h5py

```
sport_train = sport_model.fit(train_X, train_label, batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(valid_X, valid_label))
```

```
Epoch 1/20
267/267 [==============================] - 26s 84ms/step - loss: 1.5949 - accuracy: 0.2465 - val_loss: 1.5314 - val_accuracy: 0.3059
Epoch 2/20
267/267 [==============================] - 24s 90ms/step - loss: 1.4906 - accuracy: 0.3653 - val_loss: 1.3574 - val_accuracy: 0.6076
Epoch 3/20
267/267 [==============================] - 24s 90ms/step - loss: 1.3459 - accuracy: 0.4568 - val_loss: 1.1450 - val_accuracy: 0.8060
Epoch 4/20
267/267 [==============================] - 25s 94ms/step - loss: 1.1889 - accuracy: 0.5523 - val_loss: 0.9309 - val_accuracy: 0.8590
Epoch 5/20
267/267 [==============================] - 25s 93ms/step - loss: 1.0401 - accuracy: 0.6229 - val_loss: 0.7519 - val_accuracy: 0.8689
Epoch 6/20
267/267 [==============================] - 25s 93ms/step - loss: 0.9035 - accuracy: 0.6775 - val_loss: 0.6177 - val_accuracy: 0.8853
Epoch 7/20
267/267 [==============================] - 25s 92ms/step - loss: 0.8026 - accuracy: 0.7249 - val_loss: 0.5047 - val_accuracy: 0.8930
Epoch 8/20
267/267 [==============================] - 33s 124ms/step - loss: 0.7143 - accuracy: 0.7545 - val_loss: 0.4237 - val_accuracy: 0.9202
Epoch 9/20
267/267 [==============================] - 25s 95ms/step - loss: 0.6496 - accuracy: 0.7834 - val_loss: 0.3713 - val_accuracy: 0.9308
Epoch 10/20
267/267 [==============================] - 24s 90ms/step - loss: 0.5919 - accuracy: 0.8054 - val_loss: 0.3213 - val_accuracy: 0.9395
Epoch 11/20
267/267 [==============================] - 24s 91ms/step - loss: 0.5370 - accuracy: 0.8246 - val_loss: 0.2851 - val_accuracy: 0.9491
Epoch 12/20
267/267 [==============================] - 25s 92ms/step - loss: 0.5015 - accuracy: 0.8397 - val_loss: 0.2508 - val_accuracy: 0.9604
Epoch 13/20
267/267 [==============================] - 26s 99ms/step - loss: 0.4666 - accuracy: 0.8543 - val_loss: 0.2218 - val_accuracy: 0.9700
Epoch 14/20
267/267 [==============================] - 39s 145ms/step - loss: 0.4350 - accuracy: 0.8645 - val_loss: 0.2000 - val_accuracy: 0.9726
Epoch 15/20
267/267 [==============================] - 33s 123ms/step - loss: 0.4038 - accuracy: 0.8792 - val_loss: 0.1792 - val_accuracy: 0.9733
Epoch 16/20
267/267 [==============================] - 27s 103ms/step - loss: 0.3792 - accuracy: 0.8863 - val_loss: 0.1651 - val_accuracy: 0.9765
Epoch 17/20
267/267 [==============================] - 26s 97ms/step - loss: 0.3604 - accuracy: 0.8962 - val_loss: 0.1478 - val_accuracy: 0.9789
Epoch 18/20
267/267 [==============================] - 28s 103ms/step - loss: 0.3328 - accuracy: 0.9051 - val_loss: 0.1324 - val_accuracy: 0.9808
Epoch 19/20
267/267 [==============================] - 26s 99ms/step - loss: 0.3180 - accuracy: 0.9086 - val_loss: 0.1228 - val_accuracy: 0.9845
Epoch 20/20
267/267 [==============================] - 29s 110ms/step - loss: 0.3022 - accuracy: 0.9141 - val_loss: 0.1134 - val_accuracy: 0.9845
```

```
# guardamos la red, para reutilizarla en el futuro, sin tener que volver a entrenar
sport_model.save("/Users/karla/Documents/inteligencia_artificial/tareas/celulas.h5py")
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op while saving (showing 1 of 1). These functions will not be directly callable after loading.
INFO:tensorflow:Assets written to: /Users/karla/Documents/inteligencia_artificial/tareas/celulas.h5py\assets
INFO:tensorflow:Assets written to: /Users/karla/Documents/inteligencia_artificial/tareas/celulas.h5py\assets
```

Evaluamos la red

Esta parte del código evalúa la red neuronal convolucional en el conjunto de prueba utilizando el modelo de evaluate() y después se imprime la perdida y la precisión obtenida del conjunto de prueba. Posteriormente se imprime las graficas de dicha evaluación

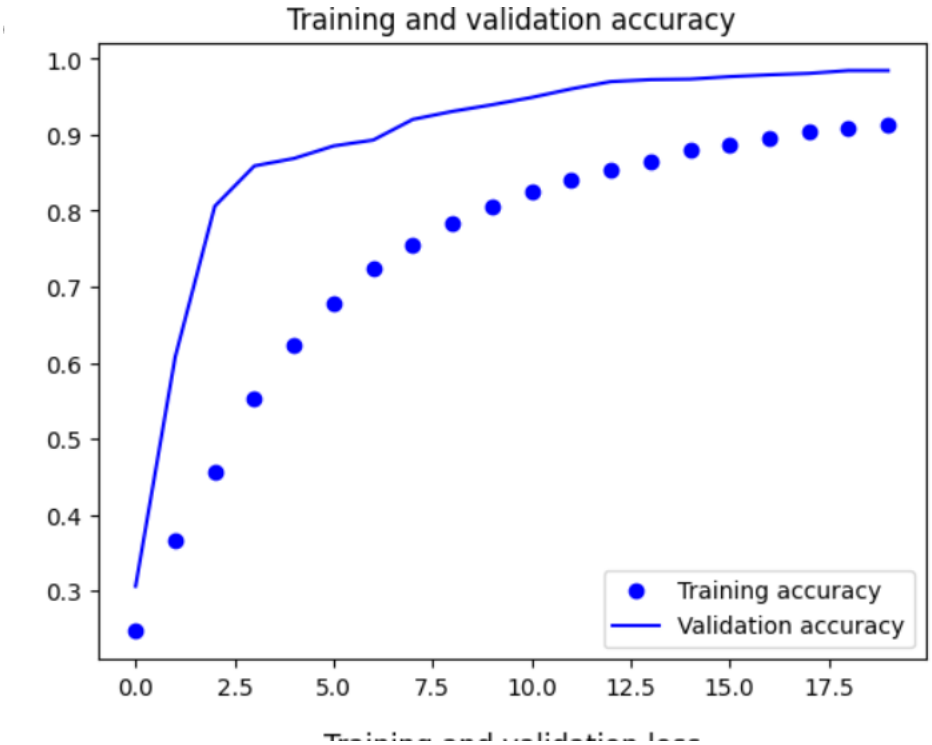
```
test_eval = sport_model.evaluate(test_X, test_Y_one_hot, verbose=1)
print('Test loss:', test_eval[0])
print('Test accuracy:', test_eval[1])
sport_train.history
```

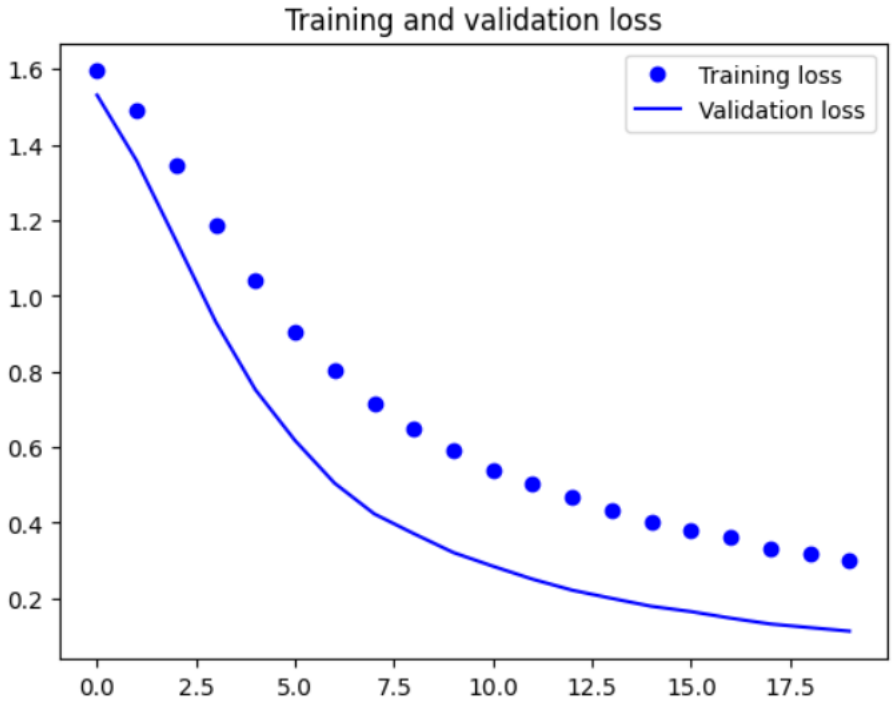
```
167/167 [==============================] - 3s 19ms/step - loss: 0.1180 - accuracy: 0.9840
```

```
Test loss: 0.11797426640987396
Test accuracy: 0.9840495586395264
```

```
    0.4568328559398651,
    0.5522580742835999,
    0.6229325532913208,
    0.67753666639328,
    0.7248680591583252,
    0.7545454502105713,
    0.7834017872810364,
    0.8053959012031555,
    0.8246334195137024,
    0.8397067189216614,
    0.8543108701705933,
    0.8645161390304565,
    0.8791788816452026,
    0.8863343000411987,
    0.8962463140487671,
    0.9051026105880737,
    0.9085630774497986,
    0.9141349196434021],
   'val_loss': [1.5314255952835083,
    1.3574124574661255,
    1.1450103521347046,
    0.9309295415878296,
    0.7518823742866516,
    0.617681205272674
```

```
accuracy = sport_train.history['accuracy']
val_accuracy = sport_train.history['val_accuracy']
loss = sport_train.history['loss']
val_loss = sport_train.history['val_loss']
epochs = range(len(accuracy))
plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```






```
predicted_classes2 = sport_model.predict(test_X)
predicted_classes=[]
for predicted_sport in predicted_classes2:
    predicted_classes.append(predicted_sport.tolist().index(max(predicted_sport)))
predicted_classes=np.array(predicted_classes)
predicted_classes.shape, test_Y.shape
```

```
167/167 [==============================] - 4s 19ms/step
```

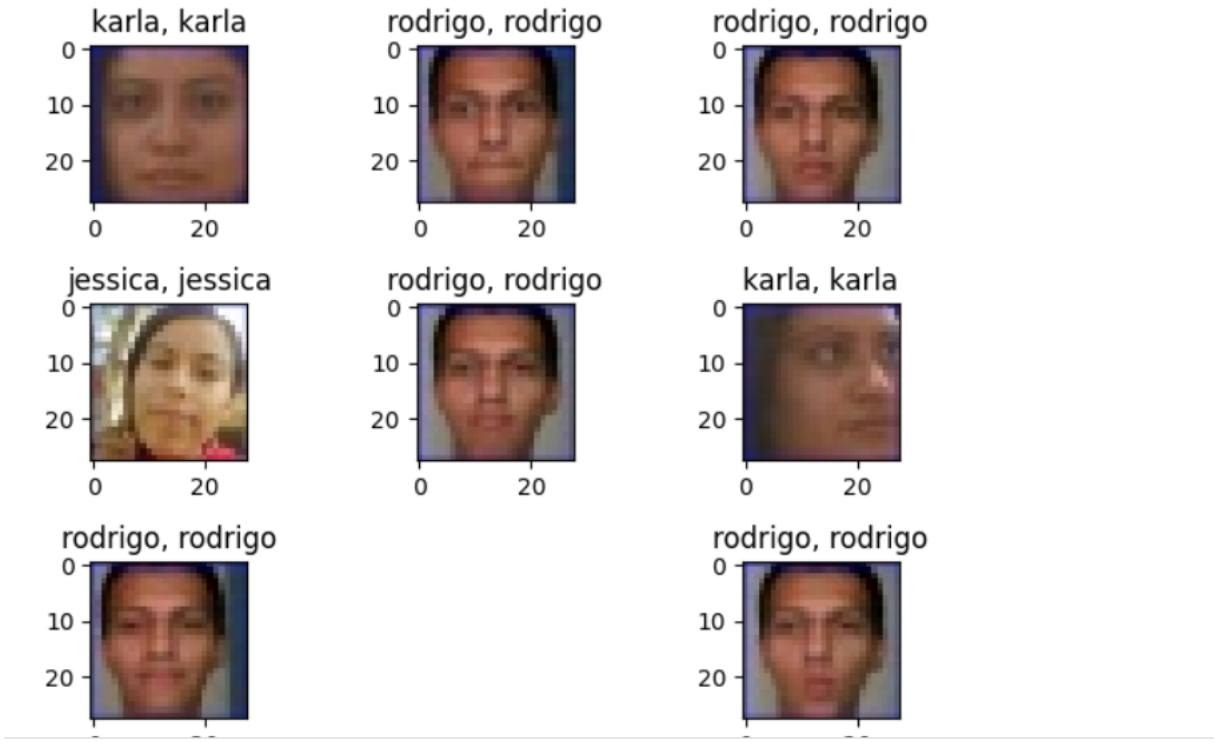
```
((5329,), (5329,))
```

Aprendamos de los errores: Qué mejorar

Aqui código busca y muestra imágenes que han sido clasificadas correctamente por el modelo en el conjunto de prueba. Primero, se busca las ubicaciones donde las clases predichas coinciden con las clases reales en el conjunto de prueba. Las imágenes se muestran en escala de grises y se les asigna un título que indica la clase predicha y la clase real de cada imagen. Esto proporciona una muestra visual de algunas de las clasificaciones correctas realizadas por el modelo en el conjunto de prueba y se realiza lo mismo para la parte incorrecta de las imagenes con imagenes incorrectamente clasificadas.

```
correct = np.where(predicted_classes==test_Y)[0]
print("Found %d correct labels" % len(correct))
for i, correct in enumerate(correct[0:9]):
    plt.subplot(3,3,i+1)
    plt.imshow(test_X[correct].reshape(28,28,3), cmap='gray', interpolation='none')
    plt.title("{}, {}".format(caras[predicted_classes[correct]],
                                                caras[test_Y[correct]]))

    plt.tight_layout()
```

```
Found 4176 correct labels
C:\Users\karla\AppData\Local\Temp\ipykernel_14560\166765134.py:4: MatplotlibDeprecationWarni
  plt.subplot(3,3,i+1)
```




```
incorrect = np.where(predicted_classes!=test_Y)[0]
print("Found %d incorrect labels" % len(incorrect))
```

```
for i, incorrect in enumerate(incorrect[0:9]):
    plt.subplot(3,3,i+1)
    plt.imshow(test_X[incorrect].reshape(28,28,3), cmap='gray', interpolation='none')
    plt.title("{}, {}".format(caras[predicted_classes[incorrect]],
                                            caras[test_Y[incorrect]]))
    plt.tight_layout()
```

```
C:\Users\karla\AppData\Local\Temp\ipykernel_9668\3252689814.py:4: MatplotlibDepre
  plt.subplot(3,3,i+1)
```


```
target_names = ["Class {}".format(i) for i in range(nClasses)]
print(classification_report(test_Y, predicted_classes, target_names=target_names))
```

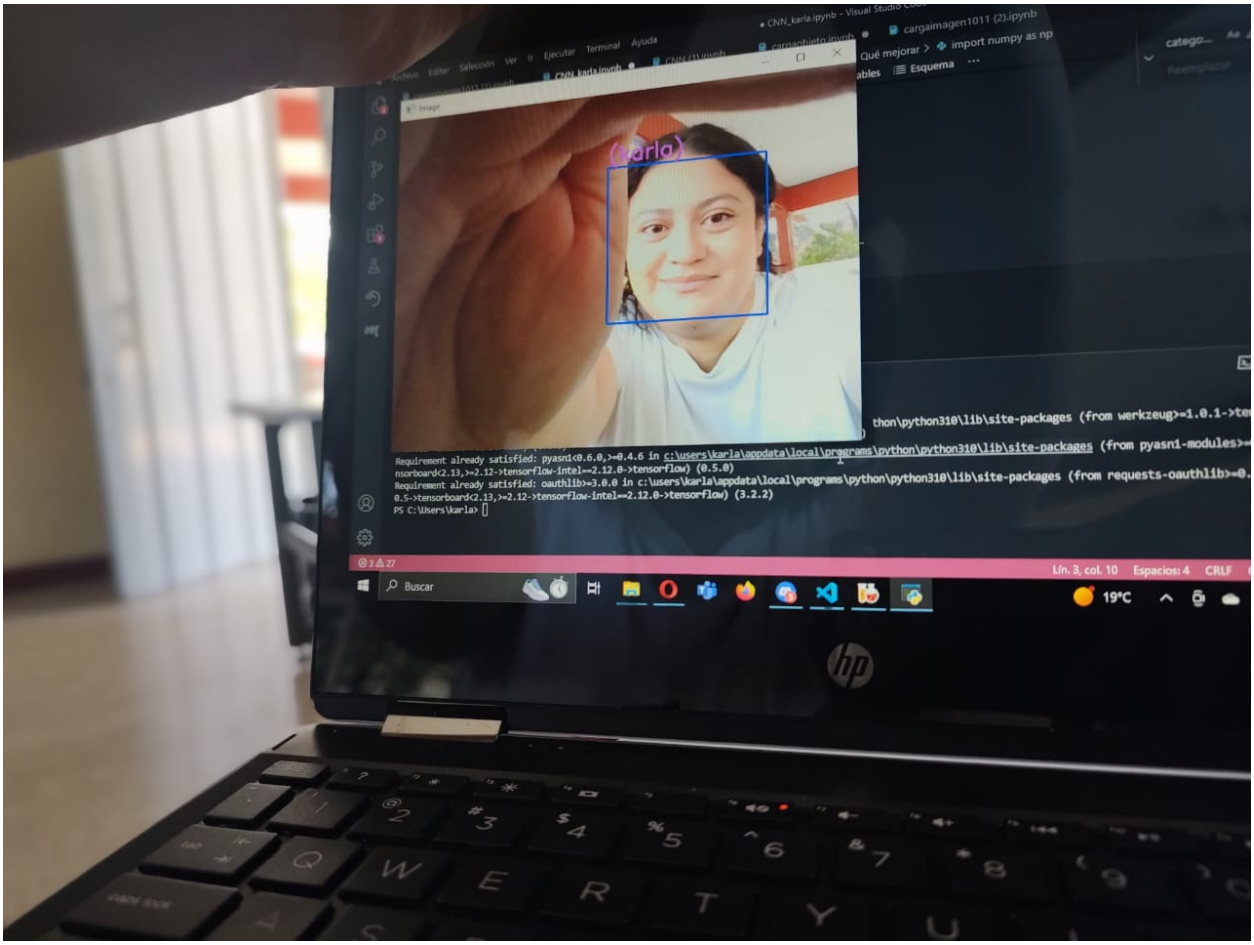
	precision	recall	f1-score	support
Class 0	0.99	1.00	0.99	1250
Class 1	1.00	0.95	0.98	1044
Class 2	0.95	1.00	0.97	1029
Class 3	0.99	0.99	0.99	1058
Class 4	0.99	0.99	0.99	948
accuracy			0.98	5329
macro avg	0.98	0.98	0.98	5329
weighted avg	0.98	0.98	0.98	5329

Probamos con la camara el modelo

Y por ultimo lo probamos con una camara para probar el modelo utilizando una biblioteca de open cv junto con un modelo previamente entrenado, para hacer la clasificación en tiempo real de rostros detectados con la camara. Básicamente, se detectan

los rostros en el video de la cámara, se recortan y se redimensionan para que coincidan con las dimensiones requeridas por el modelo. Luego, se utiliza el modelo para predecir la clase de cada rostro. Las etiquetas de clase correspondientes se muestran en la pantalla junto a los rostros detectados esto permite identificar en tiempo real a quién pertenece cada rostro capturado por la cámara

```
from PIL import Image
import numpy as np
import cv2
import pickle
from tensorflow.keras.models import load_model

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_alt.xml')

screen_width = 480
screen_height = 260

image_width = 28
image_height = 28

model = load_model('/Users/karla/Documents/inteligencia_artificial/tareas/celulas.h5py')

class_labels = ['jessica', 'rodrigo', 'william', 'karla', 'jorge']

stream = cv2.VideoCapture(0)

while True:
    (grabbed, frame) = stream.read()
    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    faces = face_cascade.detectMultiScale(rgb, scaleFactor=1.3, minNeighbors=5)

    for (x, y, w, h) in faces:
        roi_rgb = rgb[y:y + h, x:x + w]

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

        # Redimensionar la imagen
        size = (image_width, image_height)
        resized_image = cv2.resize(roi_rgb, size)
        image_array = np.array(resized_image, "uint8")
        img = image_array.reshape(1, image_width, image_height, 3)
        img = img.astype('float32')
        img /= 255

        # Realizar la predicción de la imagen
        predicted_prob = model.predict(img)

        # Obtener la clase predicha
        predicted_class = np.argmax(predicted_prob)
        class_label = class_labels[predicted_class]

        # Mostrar la etiqueta
        font = cv2.FONT_HERSHEY_SIMPLEX
        color = (255, 0, 255)
        stroke = 2
        cv2.putText(frame, f'({class_label})', (x, y - 8), font, 1, color, stroke, cv2.LINE_AA)

    cv2.imshow("Image", frame)
    key = cv2.waitKey(1) & 0xFF
    if key == ord("q"):
        break

stream.release()
cv2.waitKey(1)
cv2.destroyAllWindows()
cv2.waitKey(1)
```


(karla)
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in c:\users\karla\appdata\local\programs\python\python310\lib\site-packages (from pyasn1-modules>=
nsorboard<2.13,>=2.12->tensorflow-intel==2.12.0->tensorflow) (0.5.0)
Requirement already satisfied: oauthlib>=3.0.0 in c:\users\karla\appdata\local\programs\python\python310\lib\site-packages (from requests-oauthlib>=0.
0.5->tensorboard<2.13,>=2.12->tensorflow-intel==2.12.0->tensorflow) (3.2.2)
PS C:\Users\karla>
Buscar
Lín. 3, col. 10
Espacios: 4
CRLF
19°C