UNIVERSIDAD DE GUADALAJARA

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS

Ingenieria en Computacion Reporte de práctica

Nombre del alumno:	Miguel Angel Castro Castro
Profesor:	Erasmo Gabriel Martínez Soltero
Título de la práctica:	“Tarea 6. Convolucionales con el dataset de bueno o malo”
Fecha:	08 de Noviembre de 2022

Introducción

Todos los datos de este conjunto de datos se recopilaron de bases de datos o sitios web de acceso PÚBLICO. Este conjunto de datos consta de 2 clases, sabrosos y desagradables. La clase desagradable está poblada de magos faciales de delincuentes convictos. La clase sabrosa está poblada con imágenes faciales de personas çomunes”. De acuerdo, algunas personas çomunes”pueden ser delincuentes condenados, pero espero que el porcentaje sea muy bajo. Todas las imágenes descargadas fueron procesadas por un detector de imágenes duplicadas personalizado antes de dividirse en un conjunto de tren, un conjunto de validación y un conjunto de prueba. Esto está destinado a evitar que las imágenes sean comunes entre estos conjuntos de datos. Todas las imágenes fueron recortadas de la imagen descargada original a solo una imagen facial utilizando el módulo de recorte MTCNN. El recorte es tal que se incluye muy poco fondo extraño en la imagen recortada. Esto es para evitar que el clasificador CNN extraiga características de fondo no relevantes para la tarea de clasificación de una imagen facial. El juego de trenes tiene 5610 imágenes en la clase sabrosa y 5610 imágenes en la clase desagradable. El conjunto de prueba tiene 300 imágenes en la clase sabrosa y 300 imágenes en la clase desagradable al igual que el conjunto de validación.

Metodología

Codigo utilizado:

```python
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
import cv2
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, Activation,Dropout,Conv2D, MaxPooling2D,BatchNormalization
from tensorflow.keras.optimizers import Adam, Adamax
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras import regularizers
from tensorflow.keras.models import Model
# pprevent annoying tensorflow warning
import logging
logging.getLogger("tensorflow").setLevel(logging.ERROR)
import warnings
warnings.simplefilter("ignore")

from sklearn import metrics

from tensorflow.keras.utils import plot_model

df=pd.read_csv(r'../input/good-guysbad-guys-image-data-set/character.csv')
datasets=df.groupby('data set')
train_df=datasets.get_group('train')
test_df=datasets.get_group('test')
valid_df=datasets.get_group('valid')
# change the filepaths column to be the full path to an image
train_df['filepaths']=train_df['filepaths'].apply(lambda x: os.path.join(r'../input/good-guysbad-guys-image-data-set/', x))
test_df['filepaths']=test_df['filepaths'].apply(lambda x: os.path.join(r'../input/good-guysbad-guys-image-data-set/', x))
valid_df['filepaths']=valid_df['filepaths'].apply(lambda x: os.path.join(r'../input/good-guysbad-guys-image-data-set/', x))
# print out an imagepath to check
print(train_df['filepaths'].iloc[0])
print('train_df length: ', len(train_df), '  test_df length: ',len(test_df), '  valid_df length: ', len(valid_df))
# check the class sample balance of the train_df dataframe
print(train_df['labels'].value_counts())

# train_df is balanced, Lets look at about 200 training images and get the average height, width and aspect ratio
#to use to select an image size for the model
savory_train_dir=r'../input/good-guysbad-guys-image-data-set/train/savory' # select one of the class directories in train set
flist=sorted(os.listdir(savory_train_dir)) # get a list of the files
```

```python
ht, wt, file_count = 0,0,200
plt.figure(figsize=(20,20))
for i, f in enumerate(flist): # iterate through the files
    if i >=file_count:
        break  # only lok at first file_count number of files to save time
    else: #As long as we are here might as well show 20 training images
        imgpath=os.path.join(savory_train_dir,f)
        img=plt.imread(imgpath)
        h,w,c=np.shape(img) # use numpy shape returns height, width
        ht += h
        wt += w
        if i < 20:
            plt.subplot(4,5,i+1)
            plt.axis('off')
            plt.title(f, color='blue', fontsize=16)
            plt.imshow(img)
plt.show()
ave_h=int(ht/file_count)
ave_w=int(wt/file_count)
aspect_ratio=ave_h/ave_w
# select an image size for the model 300 pixel height should be adequate to capture image features
img_size=(300, int(300/aspect_ratio))
img_shape=(img_size[0], img_size[1], 3)
print('Average height= ', ave_h, '  average width= ', ave_w, ' average aspect ratio h/w= ', aspect_ratio, ' image shape: ', img_shape)

# create train, test and valid generators
batch_size=30 # We will use and EfficientetB3 model, with image size of (300,233) this size should not cause resource error
trgen=ImageDataGenerator(horizontal_flip=True)
t_and_v_gen=ImageDataGenerator()
train_gen=trgen.flow_from_dataframe(train_df, x_col='filepaths', y_col='labels', target_size=img_size,
                                  class_mode='categorical', color_mode='rgb', shuffle=True, batch_size=batch_size)
valid_gen=t_and_v_gen.flow_from_dataframe(valid_df, x_col='filepaths', y_col='labels', target_size=img_size,
                                  class_mode='categorical', color_mode='rgb', shuffle=False, batch_size=batch_size)
# for the test_gen we want to calculate the batch size and test steps such that batch_size X test_steps= number of samples in test set
# this insures that we go through all the sample in the test set exactly once.
length=len(test_df)
test_batch_size=sorted([int(length/n) for n in range(1,length+1) if length % n ==0 and length/n<=80],reverse=True)[0]
test_steps=int(length/test_batch_size)
test_gen=t_and_v_gen.flow_from_dataframe(test_df, x_col='filepaths', y_col='labels', target_size=img_size,
                                  class_mode='categorical', color_mode='rgb', shuffle=False, batch_size=test_batch_size)
# from the generator we can get information we will need later
```

```python
classes=list(train_gen.class_indices.keys())
class_indices=list(train_gen.class_indices.values())
class_count=len(classes)
labels=test_gen.labels
print ( 'test batch size: ' ,test_batch_size, '  test steps: ', test_steps, ' number of classes : ', class_count)
print ('{0:^12s}{1:^12s}'.format('class name', 'class index'))
for klass, index in zip(classes, class_indices):
    print(f'{klass:^12s}{str(index):^12s}')

from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers import BatchNormalization

model0 = Sequential()

model0.add(Conv2D(filters=64, kernel_size = (5,5), input_shape = img_shape, activation = 'relu'))
model0.add(BatchNormalization(axis=3))
model0.add(Conv2D(filters=64, kernel_size=(5, 5), activation='relu'))
model0.add(MaxPooling2D((2, 2)))
model0.add(BatchNormalization(axis=3))
model0.add(Dropout(0.1))

#model0.add(Conv2D(filters=128, kernel_size=(5, 5), activation='relu'))
#model0.add(BatchNormalization(axis=3))
#model0.add(Conv2D(filters=128, kernel_size=(5, 5), activation='relu'))
#model0.add(MaxPooling2D((2, 2)))
#model0.add(BatchNormalization(axis=3))
#model0.add(Dropout(0.1))

model0.add(Conv2D(filters=256, kernel_size=(5, 5), activation='relu'))
model0.add(BatchNormalization(axis=3))
model0.add(Conv2D(filters=256, kernel_size=(5, 5), activation='relu'))
model0.add(MaxPooling2D((2, 2)))
model0.add(BatchNormalization(axis=3))
model0.add(Dropout(0.1))

model0.add(Flatten())
```

```python
model0.add(Dense(256, activation='relu'))
model0.add(BatchNormalization())
model0.add(Dropout(0.5))

#model0.add(Dense(256, activation='relu'))
#model0.add(BatchNormalization())
#model0.add(Dropout(0.5))

model0.add(Dense(2, activation='softmax'))

model0.summary()

plot_model(model0,show_shapes=True,show_layer_names=False)

model0.compile(Adamax(learning_rate = 1e-3),loss='categorical_crossentropy', metrics=['accuracy'])

# create 2 useful callbacks, one to control the learning rate, and one to control early stopping based on validation loss
rlronp0=tf.keras.callbacks.ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=2,verbose=1)
estop0=tf.keras.callbacks.EarlyStopping(monitor="val_loss", patience=4, verbose=1,restore_best_weights=True)
callbacks0=[rlronp0, estop0]

epochs=20
cnn_history=model0.fit(x=train_gen,  epochs=epochs, verbose=1, callbacks=callbacks0,  validation_data=valid_gen,
               validation_steps=None,  shuffle=False,  initial_epoch=0)

def tr_plot(tr_data, start_epoch):
    #Plot the training and validation data
    tacc=tr_data.history['accuracy']
    tloss=tr_data.history['loss']
    vacc=tr_data.history['val_accuracy']
    vloss=tr_data.history['val_loss']
    Epoch_count=len(tacc)+ start_epoch
    Epochs=[]
    for i in range (start_epoch ,Epoch_count):
        Epochs.append(i+1)
    index_loss=np.argmin(vloss)#  this is the epoch with the lowest validation loss
    val_lowest=vloss[index_loss]
    index_acc=np.argmax(vacc)
    acc_highest=vacc[index_acc]
    plt.style.use('fivethirtyeight')
    sc_label='best epoch= '+ str(index_loss+1 +start_epoch)
    vc_label='best epoch= '+ str(index_acc + 1+ start_epoch)
    fig,axes=plt.subplots(nrows=1, ncols=2, figsize=(20,8))
```

```
    axes[0].plot(Epochs,tloss, 'r', label='Training loss')
    axes[0].plot(Epochs,vloss,'g',label='Validation loss' )
    axes[0].scatter(index_loss+1 +start_epoch,val_lowest, s=150, c= 'blue', label=sc_label)
    axes[0].set_title('Training and Validation Loss')
    axes[0].set_xlabel('Epochs')
    axes[0].set_ylabel('Loss')
    axes[0].legend()
    axes[1].plot (Epochs,tacc,'r',label= 'Training Accuracy')
    axes[1].plot (Epochs,vacc,'g',label= 'Validation Accuracy')
    axes[1].scatter(index_acc+1 +start_epoch,acc_highest, s=150, c= 'blue', label=vc_label)
    axes[1].set_title('Training and Validation Accuracy')
    axes[1].set_xlabel('Epochs')
    axes[1].set_ylabel('Accuracy')
    axes[1].legend()
    plt.tight_layout
    plt.show()

    tr_plot(cnn_history, 0)

    # use the trained model to make predictions and calculate accuracy since this is binary classification accuracy is sufficient
preds0=model0.predict(test_gen, steps=test_steps, verbose=1)
preds0_labels = []
errors0=0
for i, p in enumerate(preds0):
    index=np.argmax(p)
    if class_indices[index] != labels[i]:
        errors0 +=1
    preds0_labels.append(class_indices[index])
acc0= (1.0-errors0/len(preds0)) * 100
print(f'There were {errors0} errors in {len(preds0)} trials for an accuracy of  {acc0:6.2f} %')

print(metrics.classification_report(labels, preds0_labels,digits = 4))

# create an efficientNetB3 model to use for transfer learning
img_shape=(img_size[0], img_size[1], 3)
model_name='EfficientNetB3'
base_model=tf.keras.applications.efficientnet.EfficientNetB3(include_top=False, weights="imagenet",input_shape=img_shape, pooling='max')
# Note you are always told NOT to make the base model trainable initially- that is WRONG you get better results leaving it trainable
base_model.trainable=True
x=base_model.output
x=BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001 )(x)
x = Dense(1024, kernel_regularizer = regularizers.l2(l = 0.016),activity_regularizer=regularizers.l1(0.006),
```

```
x=Dropout(rate=.3, seed=123)(x)
x = Dense(128, kernel_regularizer = regularizers.l2(l = 0.016),activity_regularizer=regularizers.l1(0.006),
                bias_regularizer=regularizers.l1(0.006) ,activation='relu')(x)
x=Dropout(rate=.45, seed=123)(x)
output=Dense(class_count, activation='softmax')(x)
model=Model(inputs=base_model.input, outputs=output)
lr=.001 # start with this learning rate
model.compile(Adamax(learning_rate=lr), loss='categorical_crossentropy', metrics=['accuracy'])

base_model.summary()

plot_model(model,show_shapes=True,show_layer_names=False)

# create 2 useful callbacks, one to control the learning rate, and one to control early stopping based on validation loss
rlronp=tf.keras.callbacks.ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=2,verbose=1)
estop=tf.keras.callbacks.EarlyStopping(monitor="val_loss", patience=4, verbose=1,restore_best_weights=True)
callbacks=[rlronp, estop]

# train the model
epochs=20
history=model.fit(x=train_gen,  epochs=epochs, verbose=1, callbacks=callbacks,  validation_data=valid_gen,
               validation_steps=None,  shuffle=False,  initial_epoch=0)

tr_plot(history,0)

# use the trained model to make predictions and calculate accuracy since this is binary classification accuracy is sufficient
preds=model.predict(test_gen, steps=test_steps, verbose=1)
preds_labels = []
errors=0
for i, p in enumerate(preds):
    index=np.argmax(p)
    if class_indices[index] != labels[i]:
        errors +=1
    preds_labels.append(class_indices[index])
acc= (1.0-errors/len(preds)) * 100
print(f'There were {errors} errors in {len(preds)} trials for an accuracy of  {acc:6.2f} %')

print(metrics.classification_report(labels, preds_labels,digits = 4))

#lets show the images in the images to predict directory
plt.figure(figsize=(20,5))
```

```python
plt.figure(figsize=(20,5))
sdir=r'../input/good-guysbad-guys-image-data-set/images to predict'
flist=sorted(os.listdir(sdir))
for i,f in enumerate(flist):
    fpath=os.path.join(sdir,f)
    img=plt.imread(fpath)
    plt.subplot(2,6,i +1)
    plt.axis('off')
    plt.title(f, color='blue', fontsize=16)
    plt.imshow(img)
plt.show()

def predictor(sdir,img_size, average=False):
    img_list=[]
    fname=[]
    class_list=[]
    savory_probs=[]
    unsavory_probs=[]
    prob_list=[]
    flist=sorted(os.listdir(sdir))
    img_count=len(flist)
    for f in flist:
        fpath=os.path.join(sdir,f)
        img=plt.imread(fpath)
        img=cv2.resize(img, (img_size[1], img_size[0]))
        img_list.append(img)
        fname.append(f)
    img_array=np.array(img_list)
    preds=model.predict(img_array, steps=img_count)
    for p in preds:
        index=np.argmax(p)
        klass=classes[index]
        class_list.append(klass)
        savory_probs.append(p[0])
        unsavory_probs.append(p[1])
        prob_list.append(p[index])
    if average == False:
        Fseries=pd.Series(fname, name='file name')
        Lseries=pd.Series(class_list, name='Predicted Class')
        Pseries=pd.Series(prob_list, name='Probability')
        df=pd.concat([Fseries, Lseries, Pseries], axis=1)
        print(df.head(img_count))
        return df
```

```python
    else: #average the probabilities
        savory_sum=0
        unsavory_sum=0
        for savory, unsavory in zip(savory_probs, unsavory_probs):
            savory_sum += savory/img_count
            unsavory_sum += unsavory/img_count
        if savory_sum>= unsavory_sum:
            klass='Savory'
            prob=savory_sum
        else:
            klass='Unsavory'
            prob=unsavory_sum
            return klass
    print(f' majority class is {klass} with probability {prob:6.2f}')

# call the predictor on the images to predict directory. It has 10 images (01 to 10) of Dr Fauci and 1(11) image of a felon.
#So 10 of the predictions such be savory and 1 should be unsavory
sdir= r'../input/good-guysbad-guys-image-data-set/images to predict'
result=predictor(sdir,img_size, average=False)

import tensorflow_addons as tfa
from tensorflow.keras import layers
from vit_keras import vit

# create train, test and valid generators
img_size2 = (288,224)
#img_size2 = (300,240)
batch_size=30 # We will use and EfficientetB3 model, with image size of (300,233) this size should not cause resource error
trgen=ImageDataGenerator(horizontal_flip=True)
t_and_v_gen=ImageDataGenerator()
train_gen2=trgen.flow_from_dataframe(train_df, x_col='filepaths', y_col='labels', target_size=img_size2,
                                   class_mode='categorical', color_mode='rgb', shuffle=True, batch_size=batch_size)
valid_gen2=t_and_v_gen.flow_from_dataframe(valid_df, x_col='filepaths', y_col='labels', target_size=img_size2,
                                   class_mode='categorical', color_mode='rgb', shuffle=False, batch_size=batch_size)
# for the test_gen we want to calculate the batch size and test steps such that batch_size X test_steps= number of samples in test set
# this insures that we go through all the sample in the test set exactly once.
length=len(test_df)
test_batch_size=sorted([int(length/n) for n in range(1,length+1) if length % n ==0 and length/n<=80],reverse=True)[0]
test_steps=int(length/test_batch_size)
test_gen2=t_and_v_gen.flow_from_dataframe(test_df, x_col='filepaths', y_col='labels', target_size=img_size2,
                                   class_mode='categorical', color_mode='rgb', shuffle=False, batch_size=test_batch_size)
```

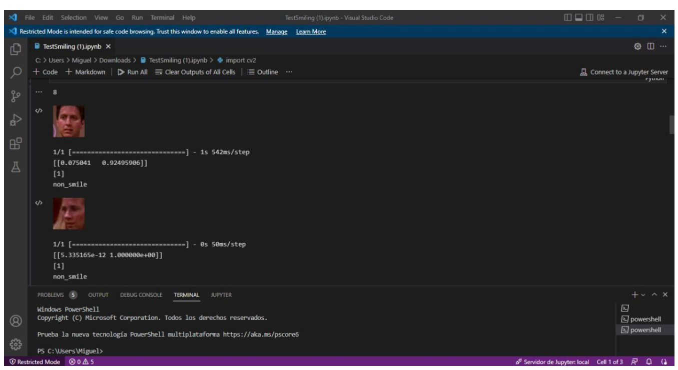
```python
classes=list(train_gen2.class_indices.keys())
class_indices=list(train_gen2.class_indices.values())
class_count=len(classes)
labels=test_gen2.labels
print ( 'test batch size: ' ,test_batch_size, '  test steps: ', test_steps, ' number of classes : ', class_count)
print ('{0:^12s}{1:^12s}'.format('class name', 'class index'))
for klass, index in zip(classes, class_indices):
    print(f'{klass:^12s}{str(index):^12s}')

vit_model = vit.vit_b16(image_size = (288,224),
                        activation = 'softmax',
                        pretrained = True,
                        include_top = True,
                        pretrained_top = True,
                        classes = 2)

plot_model(vit_model,show_shapes=True,show_layer_names=False)

vit_model.summary()

def build_model2():
    inputs = layers.Input(shape=(288,224,3))

    x = vit_model(inputs)
    x = BatchNormalization()(x)
    x = Dense(224, activation = tfa.activations.gelu)(x)
    x = Dense(2,activation='sigmoid', name = 'sigmoid')(x)

    model = tf.keras.Model(inputs = inputs, outputs = x)

    return model

model2 = build_model2()
model2.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
             loss = 'categorical_crossentropy',
             metrics=['accuracy'])
model2.summary()

plot_model(model2,show_shapes=True,show_layer_names=False)


ckpt = tf.keras.callbacks.ModelCheckpoint("vit_b16_weights.h5", save_best_only = True, save_weights_only = True)
```

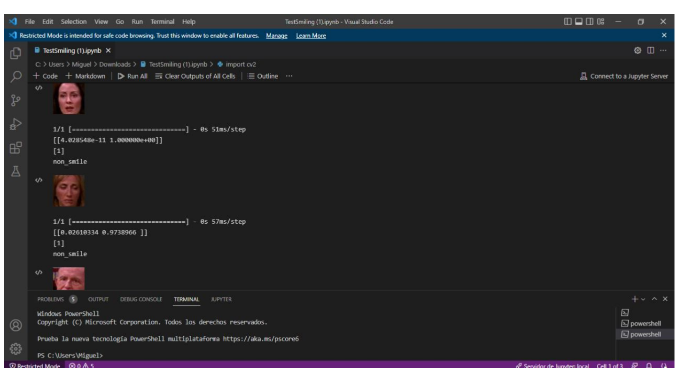
```python
model2.load_weights('vit_b16_weights.h5')

# use the trained model to make predictions and calculate accuracy since this is binary classification accuracy is sufficient
preds2=model2.predict(test_gen2, steps=test_steps, verbose=1)
preds2_labels = []
errors2=0
for i, p in enumerate(preds2):
    index=np.argmax(p)
    if class_indices[index] != labels[i]:
        errors2 +=1
    preds2_labels.append(class_indices[index])
acc2 = (1.0-errors2/len(preds2)) * 100
print(f'There were {errors2} errors in {len(preds2)} trials for an accuracy of  {acc2:6.2f} %')

print(metrics.classification_report(labels, preds2_labels,digits = 3))

tr_plot(vit_train_history,0)

# create train, test and valid generators
img_size3 = (224,224)
batch_size=30 # We will use and EfficientetB3 model, with image size of (300,233) this size should not cause resource error
trgen=ImageDataGenerator(horizontal_flip=True)
t_and_v_gen=ImageDataGenerator()
train_gen3=trgen.flow_from_dataframe(train_df, x_col='filepaths', y_col='labels', target_size=img_size3,
                                   class_mode='categorical', color_mode='rgb', shuffle=True, batch_size=batch_size)
valid_gen3=t_and_v_gen.flow_from_dataframe(valid_df, x_col='filepaths', y_col='labels', target_size=img_size3,
                                   class_mode='categorical', color_mode='rgb', shuffle=False, batch_size=batch_size)
# for the test_gen we want to calculate the batch size and test steps such that batch_size X test_steps= number of samples in test set
# this insures that we go through all the sample in the test set exactly once.
length=len(test_df)
test_batch_size=sorted([int(length/n) for n in range(1,length+1) if length % n ==0 and length/n<=80],reverse=True)[0]
test_steps=int(length/test_batch_size)
test_gen3=t_and_v_gen.flow_from_dataframe(test_df, x_col='filepaths', y_col='labels', target_size=img_size3,
                                   class_mode='categorical', color_mode='rgb', shuffle=False, batch_size=test_batch_size)
# from the generator we can get information we will need later
classes=list(train_gen3.class_indices.keys())
class_indices=list(train_gen3.class_indices.values())
class_count=len(classes)
labels=test_gen3.labels
print ( 'test batch size: ' ,test_batch_size, '  test steps: ', test_steps, ' number of classes : ', class_count)
```

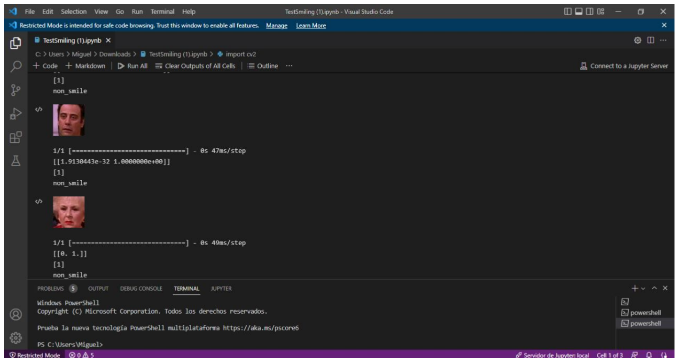
```
vgg16_model = VGG16(
    include_top=False,
    weights="imagenet",
    input_shape=(224,224,3)
)

vgg16_model.summary()

plot_model(vgg16_model,show_shapes=True,show_layer_names=False)

#freeze base layers?
for layer in vgg16_model.layers:
    layer.trainable=False

NUM_CLASSES = 2

model3 = Sequential()
model3.add(vgg16_model)
model3.add(layers.Flatten())
model3.add(layers.Dropout(0.5))
model3.add(layers.Dense(NUM_CLASSES, activation = 'sigmoid'))

model3.compile(loss='categorical_crossentropy',optimizer=Adam(learning_rate=1e-3),metrics=['accuracy'])

model3.summary()

plot_model(model3,show_shapes=True,show_layer_names=False)

EPOCHS = 20
#early_stop = EarlyStopping(monitor='accuracy',patience = 6)

#my_callbacks = [early_stop]
rlronp3=tf.keras.callbacks.ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=2,verbose=1)
estop3=tf.keras.callbacks.EarlyStopping(monitor="val_loss", patience=4, verbose=1,restore_best_weights=True)
callbacks3=[rlronp3, estop3]

vgg16_history = model3.fit(
    train_gen3,
#     steps_per_epoch=TRAIN_LEN // TRAIN_BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=valid_gen3,
```

Resultados

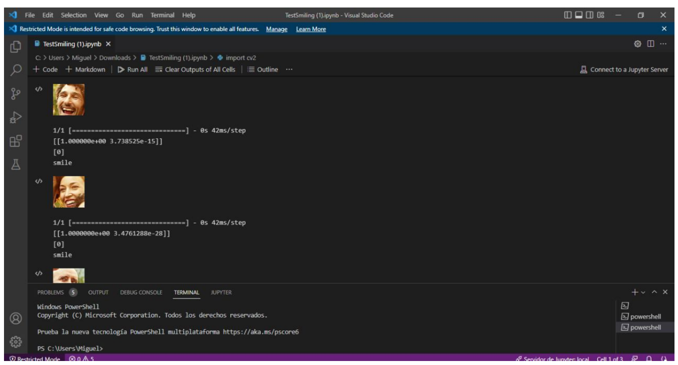
Usamos de base el de feliz y enojado



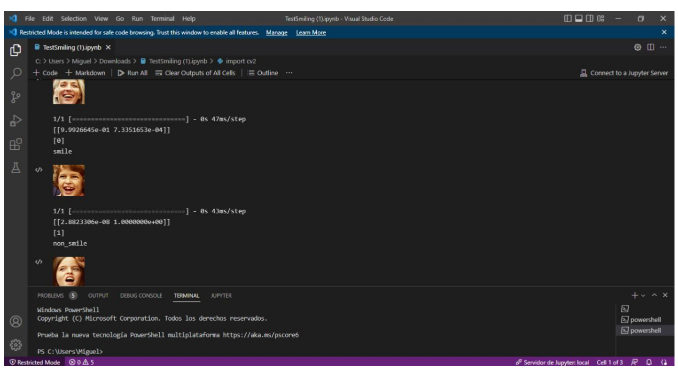




[1]
non_smile
1/1 [==============================] - 0s 47ms/step
[[1.9130443e-32 1.0000000e+00]]
[1]
non_smile
1/1 [==============================] - 0s 49ms/step
[[0. 1.]]
[1]
non_smile

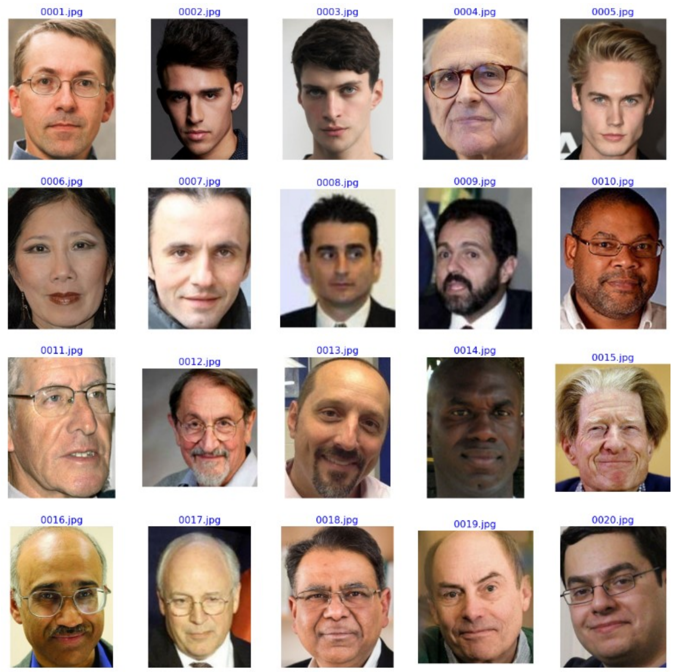


1/1 [==============================] - 0s 42ms/step
[[1.000000e+00 3.738525e-15]]
[0]
smile
1/1 [==============================] - 0s 42ms/step
[[1.0000000e+00 3.4761288e-28]]
[0]
smile



1/1 [==============================] - 0s 47ms/step
[[9.9926645e-01 7.3351653e-04]]
[0]
smile
1/1 [==============================] - 0s 43ms/step
[[2.8823306e-08 1.0000000e+00]]
[1]
non_smile

Resultados:

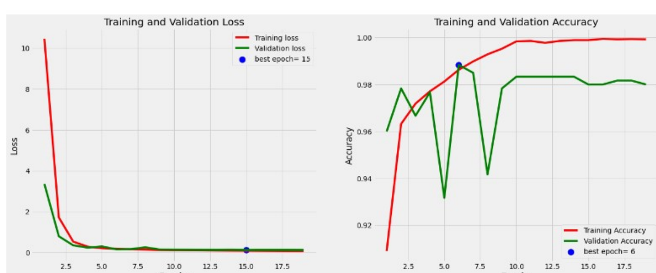



```
Found 11220 validated image filenames belonging to 2 classes.
Found 600 validated image filenames belonging to 2 classes.
Found 600 validated image filenames belonging to 2 classes.
test batch size:  75   test steps:  8  number of classes :  2
 class name class index
   savory       0
  unsavory      1
```

```
374/374 [==============================] - 242s 595ms/step - loss: 10.4381 - accuracy:
0.9090 - val_loss: 3.3489 - val_accuracy: 0.9600
Epoch 2/20
374/374 [==============================] - 219s 585ms/step - loss: 1.7156 - accuracy: 0.
9632 - val_loss: 0.7961 - val_accuracy: 0.9783
Epoch 3/20
374/374 [==============================] - 218s 583ms/step - loss: 0.5338 - accuracy: 0.
9718 - val_loss: 0.3514 - val_accuracy: 0.9667
Epoch 4/20
374/374 [==============================] - 218s 584ms/step - loss: 0.2874 - accuracy: 0.
9771 - val_loss: 0.2406 - val_accuracy: 0.9767
Epoch 5/20
374/374 [==============================] - 219s 586ms/step - loss: 0.2134 - accuracy: 0.
9813 - val_loss: 0.2990 - val_accuracy: 0.9317
Epoch 6/20
374/374 [==============================] - 219s 584ms/step - loss: 0.1807 - accuracy: 0.
9863 - val_loss: 0.1546 - val_accuracy: 0.9883
Epoch 7/20
374/374 [==============================] - 219s 584ms/step - loss: 0.1588 - accuracy: 0.
9898 - val_loss: 0.1679 - val_accuracy: 0.9850
Epoch 8/20
374/374 [==============================] - 219s 585ms/step - loss: 0.1447 - accuracy: 0.
9929 - val_loss: 0.2550 - val_accuracy: 0.9417
```

```
Epoch 00008: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
Epoch 9/20
374/374 [==============================] - 219s 586ms/step - loss: 0.1247 - accuracy: 0.
9953 - val_loss: 0.1501 - val_accuracy: 0.9783
Epoch 10/20
374/374 [==============================] - 219s 584ms/step - loss: 0.1113 - accuracy: 0.
9984 - val_loss: 0.1413 - val_accuracy: 0.9833
Epoch 11/20
374/374 [==============================] - 219s 585ms/step - loss: 0.1057 - accuracy: 0.
9986 - val_loss: 0.1369 - val_accuracy: 0.9833
Epoch 12/20
374/374 [==============================] - 219s 586ms/step - loss: 0.1023 - accuracy: 0.
9978 - val_loss: 0.1329 - val_accuracy: 0.9833
Epoch 13/20
374/374 [==============================] - 218s 582ms/step - loss: 0.0961 - accuracy: 0.
9986 - val_loss: 0.1299 - val_accuracy: 0.9833
Epoch 14/20
374/374 [==============================] - 219s 585ms/step - loss: 0.0915 - accuracy: 0.
9989 - val_loss: 0.1391 - val_accuracy: 0.9833
Epoch 15/20
374/374 [==============================] - 218s 583ms/step - loss: 0.0886 - accuracy: 0.
9989 - val_loss: 0.1242 - val_accuracy: 0.9800
Epoch 16/20
374/374 [==============================] - 218s 583ms/step - loss: 0.0845 - accuracy: 0.
9995 - val_loss: 0.1303 - val_accuracy: 0.9800
Epoch 17/20
374/374 [==============================] - 219s 585ms/step - loss: 0.0816 - accuracy: 0.
9993 - val_loss: 0.1289 - val_accuracy: 0.9817
```

```
Epoch 00017: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
Epoch 18/20
374/374 [==============================] - 219s 584ms/step - loss: 0.0785 - accuracy: 0.
9994 - val_loss: 0.1293 - val_accuracy: 0.9817
Epoch 19/20
374/374 [==============================] - 218s 583ms/step - loss: 0.0773 - accuracy: 0.
9993 - val_loss: 0.1265 - val_accuracy: 0.9800

Epoch 00019: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
Restoring model weights from the end of the best epoch.
Epoch 00019: early stopping
```



	file name	Predicted Class	Probability
0	01.jpg	savory	0.992805
1	02.jpg	savory	0.996700
2	03.jpg	savory	0.994198
3	04.jpg	savory	0.987547
4	05.jpg	savory	0.981984
5	06.jpg	savory	0.983529
6	07.jpg	savory	0.991409
7	08.jpg	savory	0.990148
8	09.jpg	savory	0.992387
9	10.jpg	savory	0.985571
10	11.jpg	unsavory	0.996345

Links de los codigos:
https://colab.research.google.com/drive/1CpVsgPjCdXXDmfICw7
bWmfXyUhh198DX?usp=sharing
https://colab.research.google.com/drive/1q44c7xr
RRQ1YA0Z7e7WBi-e-xUAutIYy?usp=sharing
https://drive.google.com/file/d/1VoM80lDfrCkrZeg
67E5p9SDgUwtNYRw1/view?usp=sharing

Conclusiones

Las Redes neuronales convolucionales son un tipo de redes neuronales artificiales donde las «neuronas» corresponden a campos receptivos de una manera muy similar a las neuronas en la corteza visual primaria (V1) de un cerebro biológico.

Referencias

https://www.kaggle.com/datasets/gpiosenka/good-guysbad-guys-image-data-set

https://www.kaggle.com/code/irenewang345/dl-project-goodguybadguy?scriptVersionId=95580795