07MIARG38_PullupaxiAtaballo_Mayra

May 22, 2022

1 07MIAR - Redes Neuronales y Deep Learning: Proyecto de programación "Deep Vision in classification tasks"

1) Estrategia 1: Entrenar desde cero o from scratch

La primera estrategia a comparar será una **red neuronal profunda** que el **alumno debe diseñar, entrenar y optimizar**. Se debe **justificar empíricamente** las decisiones que llevaron a la selección de la **arquitectura e hiperparámetros final**. Se espera que el alumno utilice todas las **técnicas de regularización** mostradas en clase de forma justificada para la mejora del rendimiento de la red neuronal (*weight regularization*, *dropout*, *batch normalization*, *data augmentation*, etc.).

1.1 CARGA DEL CONJUNTO DE DATOS

[7]:
```
!pip install --upgrade --force-reinstall --no-deps kaggle
```

```
Collecting kaggle
  Downloading kaggle-1.5.12.tar.gz (58 kB)
     |                  | 58 kB 3.0 MB/s
Building wheels for collected packages: kaggle
  Building wheel for kaggle (setup.py) … done
  Created wheel for kaggle: filename=kaggle-1.5.12-py3-none-any.whl size=73051
sha256=a88035717f8ecce1ff181e66a6e48bdf629fb89699963592218f883ce838d6ea
  Stored in directory: /root/.cache/pip/wheels/62/d6/58/5853130f941e75b2177d281e
b7e44b4a98ed46dd155f556dc5
Successfully built kaggle
Installing collected packages: kaggle
  Attempting uninstall: kaggle
    Found existing installation: kaggle 1.5.12
    Uninstalling kaggle-1.5.12:
      Successfully uninstalled kaggle-1.5.12
Successfully installed kaggle-1.5.12
```

[1]:
```
# Seleccionar el API Token personal previamente descargado (fichero kaggle.json)
from google.colab import files
files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving kaggle.json to kaggle.json
```

[1]:
```
{'kaggle.json':
b'{"username":"mayrapullupaxi","key":"577bdaf40ee5d5040f975e84762bb39e"}'}
```

[2]:
```
# Se crea un directorio en el que copiamos el fichero kaggle.json
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

[3]:
```
# Lista de los datasets disponibles en kaggle para su descarga
!kaggle datasets list
```

```
ref                                                                 title
size  lastUpdated          downloadCount  voteCount  usabilityRating
-------------------------------------------------------------------
----------------------------------------------  -----  -------------------
-------------  ---------  ----------------
muratkokludataset/date-fruit-datasets                               Date Fruit
Datasets                             408KB  2022-04-03 09:25:39          8911
1218  0.9375
victorsoeiro/netflix-tv-shows-and-movies                            Netflix TV
Shows and Movies                       2MB  2022-05-15 00:01:23          1271
52  1.0
mdmahmudulhasansuzan/students-adaptability-level-in-online-education  Students
Adaptability Level in Online Education    6KB  2022-04-16 04:46:28
5960         153  1.0
muratkokludataset/acoustic-extinguisher-fire-dataset                Acoustic
Extinguisher Fire Dataset              621KB  2022-04-02 22:59:36
1427        1059  0.9375
muratkokludataset/rice-image-dataset                                Rice Image
Dataset                              219MB  2022-04-03 02:12:00          1745
1012  0.875
muratkokludataset/raisin-dataset                                    Raisin
Dataset                                  112KB  2022-04-03 00:23:16
716         899  0.9375
muratkokludataset/dry-bean-dataset                                  Dry Bean
Dataset                                    5MB  2022-04-02 23:19:30
595         896  0.9375
paradisejoy/top-hits-spotify-from-20002019                          Top Hits
Spotify from 2000-2019                    94KB  2022-04-26 17:30:03
1913         47  1.0
muratkokludataset/pistachio-dataset                                 Pistachio
Dataset                                    2MB  2022-04-03 08:38:21           645
918  0.9375
muratkokludataset/rice-msc-dataset                                  Rice MSC
Dataset                                  102MB  2022-04-03 01:33:52
```

```
278         886  0.9375
muratkokludataset/grapevine-leaves-image-dataset                      Grapevine
Leaves Image Dataset                   109MB  2022-04-03 09:00:54          228
925  0.875
muratkokludataset/rice-dataset-commeo-and-osmancik                    Rice
Dataset Commeo and Osmancik                 524KB  2022-04-03 00:40:03
137         874  0.875
muratkokludataset/durum-wheat-dataset                                 Durum
Wheat Dataset                             983MB  2022-04-03 00:02:29
116         897  0.875
mysarahmadbhat/airline-passenger-satisfaction                         Airline
Passenger Satisfaction                    2MB  2022-05-19 11:46:02
394          16  1.0
muratkokludataset/pistachio-image-dataset                             Pistachio
Image Dataset                          27MB  2022-03-28 18:01:27          602
965  0.9375
muratkokludataset/pumpkin-seeds-dataset                               Pumpkin
Seeds Dataset                           393KB  2022-03-28 18:28:16
708         894  0.9375
ujjwalchowdhury/energy-efficiency-data-set                            Energy
Efficiency Data Set                         6KB  2022-05-12 13:51:03
579          32  0.9705882
surajjha101/stores-area-and-sales-data
Supermarket store branches sales analysis         10KB  2022-04-29 11:10:16
1731          69  1.0
rinichristy/covid19-coronavirus-pandemic                              COVID-19
Coronavirus Pandemic                       9KB  2022-04-05 08:43:16
4631         106  1.0
jjdaguirre/forbes-billionaires-2022                                   Forbes
billionaires 2022                         56KB  2022-04-30 18:48:22
866          27  1.0
```

[4]:
```
# Descargar un dataset de cierta competición
!kaggle competitions download -c dog-breed-identification
```

```
Downloading dog-breed-identification.zip to /content
100% 689M/691M [00:28<00:00, 16.5MB/s]
100% 691M/691M [00:28<00:00, 25.6MB/s]
```

[5]:
```
# Se crea directorio para descomprimir los datos
!mkdir my_dataset
```

[ ]:
```
# Se descomprime los datos y los dejamos listos para trabajar
!unzip dog-breed-identification.zip -d my_dataset
```

[7]:
```
# Ser importan las librerias a utilizar
```

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
from keras.preprocessing.image import img_to_array, load_img, ImageDataGenerator
```

[8]:
```python
# Se carga los labels a un dataframe

df_labels = pd.read_csv('my_dataset/labels.csv')
df_labels.head()
```

[8]:

	id	breed
0	000bec180eb18c7604dcecc8fe0dba07	boston_bull
1	001513dfcb2ffafc82cccf4d8bbaba97	dingo
2	001cdf01b096e06d78e9e5112d419397	pekinese
3	00214f311d5d2247d5dfe4fe24b2303d	bluetick
4	0021f9ceb3235effd7fcde7f7538ed62	golden_retriever

[9]:
```python
# Se arma el path de la imagen asociando a cada imagen dada

img_file ='my_dataset/train/'
df = df_labels.assign(id = lambda x: x['id'] +'.jpg')
df.head()
```

[9]:

	id	breed
0	000bec180eb18c7604dcecc8fe0dba07.jpg	boston_bull
1	001513dfcb2ffafc82cccf4d8bbaba97.jpg	dingo
2	001cdf01b096e06d78e9e5112d419397.jpg	pekinese
3	00214f311d5d2247d5dfe4fe24b2303d.jpg	bluetick
4	0021f9ceb3235effd7fcde7f7538ed62.jpg	golden_retriever

1.2. ACONDICIONAMIENTO DEL CONJUNTO DE DATOS

En este caso se genera dos grupos de train y valiation y se estandariza, además se toma como bathc size 32, se puede apreciar que existen 120 clases de razas de perros

[10]:
```python
data_generator = ImageDataGenerator(rescale = 1./255, validation_split = 0.2)

train_generator = data_generator.flow_from_dataframe(
dataframe = df,
directory = "my_dataset/train/",
x_col = "id",
y_col = "breed",
batch_size = 32,
seed = 42,
subset = "training",
class_mode = "categorical",
target_size = (224,224))
```

```
validation_generator = data_generator.flow_from_dataframe(
dataframe = df,
directory = "my_dataset/train/",
x_col = "id",
y_col = "breed",
batch_size = 32,
suffle = False,
seed = 42,
subset = "validation",
class_mode = "categorical",
target_size = (224,224))
```

```
Found 8178 validated image filenames belonging to 120 classes.
Found 2044 validated image filenames belonging to 120 classes.
```

[11]:

```
# Para verificar la data generada de grupo de datos, tamaño de imagen y␣
↪etiquetas

for data_batch, labels_batch in train_generator:
  print('Dimensión de los datos:',data_batch.shape)
  print('Dimensión de las etiquetas:',labels_batch.shape)
  break
```

```
Dimensión de los datos: (32, 224, 224, 3)
Dimensión de las etiquetas: (32, 120)
```

1.3. INSPECCION DEL CONJUNTO DE DATOS

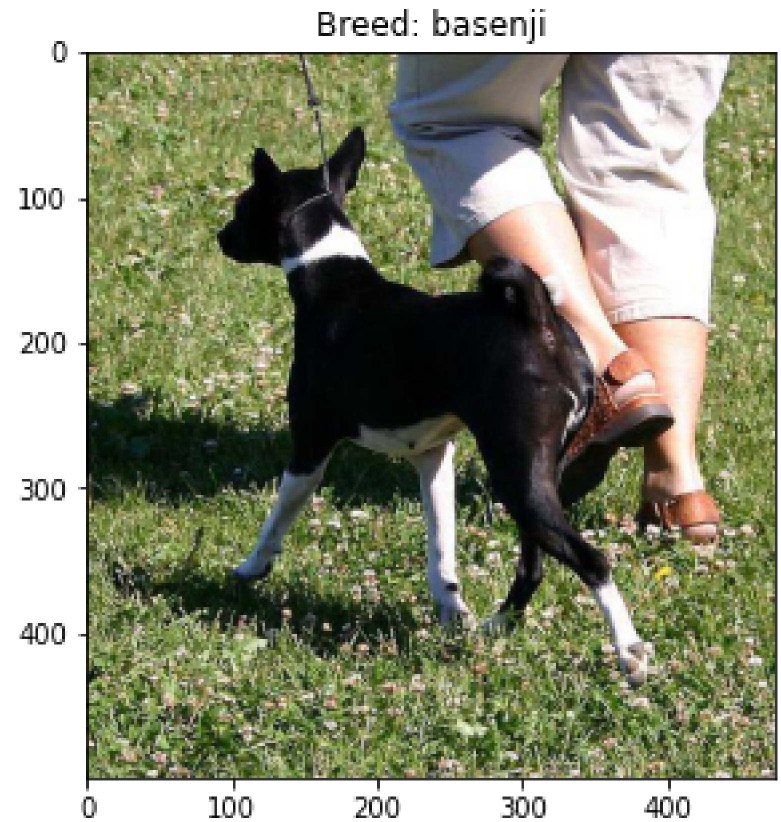
[17]:

```
# Se crea una función para visualizar los ejemplos aleatoriamente

def display_imagen(value):
  # Se selecciona la imagen
  image = cv2.imread("my_dataset/train/" + df.id.values[value])
  # Seleccionar el label
  label = df.breed.values[value]
  # Se muestra la imagen
  fig = plt.figure(figsize=(5,5))
  plt.title('Breed: %s' % label)
  img = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
  plt.imshow(img)

display_imagen(np.random.randint(0, data_batch.shape[0]))
display_imagen(np.random.randint(0, data_batch.shape[0]))
display_imagen(np.random.randint(0, data_batch.shape[0]))
```


Breed: redbone
0
100
200
300
400
0
100
200
300



1.4. DESARROLLO DE LA ARQUITECTURA DE RED NEURONAL Y ENTRENAMIENTO DE LA SOLUCION

Se construye la arquitectura de red, para ello se ha realizado cuatro capas convolucionales, con funciones de activación Relu, además se ha agragado pooling para reducir el número de parámetros

[13]:
```
from keras.models import Sequential
from keras.layers import␣
↪Dense,Dropout,Input,MaxPooling2D,ZeroPadding2D,Conv2D,Flatten
from keras.losses import categorical_crossentropy
from tensorflow.keras.optimizers import Adam

model = Sequential()

model.add(ZeroPadding2D((1,1), input_shape=(224,224,3)))
model.add(Conv2D(16, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

model.add(ZeroPadding2D(padding=(1,1)))
model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

model.add(Conv2D(128, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dense(120, activation='softmax'))

model.compile(loss = categorical_crossentropy, optimizer = 'adam', metrics =␣
↪['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 16)      448

 max_pooling2d (MaxPooling2D  (None, 111, 111, 16)     0
 )

 conv2d_1 (Conv2D)           (None, 109, 109, 32)      4640

 max_pooling2d_1 (MaxPooling  (None, 54, 54, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 52, 52, 64)        18496
```

```
 max_pooling2d_2 (MaxPooling  (None, 26, 26, 64)       0
 2D)

 conv2d_3 (Conv2D)           (None, 24, 24, 128)       73856

 max_pooling2d_3 (MaxPooling  (None, 12, 12, 128)      0
 2D)

 flatten (Flatten)           (None, 18432)             0

 dense (Dense)               (None, 512)               9437696

 dense_1 (Dense)             (None, 120)               61560

=================================================================
Total params: 9,596,696
Trainable params: 9,596,696
Non-trainable params: 0
_________________________________________________________________
```

[76]:

```
# Luego vamos a entrenarla nuestra data

H1=model.fit(train_generator,
                      steps_per_epoch = 8,
                      validation_data = validation_generator,
                      validation_steps = 64,
                      epochs = 20,
                      verbose = 1)
```

```
Epoch 1/20
8/8 [==============================] - 47s 6s/step - loss: 4.8383 - accuracy:
0.0039 - val_loss: 4.7903 - val_accuracy: 0.0078
Epoch 2/20
8/8 [==============================] - 46s 6s/step - loss: 4.7918 - accuracy:
0.0000e+00 - val_loss: 4.7953 - val_accuracy: 0.0113
Epoch 3/20
8/8 [==============================] - 46s 6s/step - loss: 4.7839 - accuracy:
0.0078 - val_loss: 4.7927 - val_accuracy: 0.0093
Epoch 4/20
8/8 [==============================] - 46s 6s/step - loss: 4.7915 - accuracy:
0.0156 - val_loss: 4.7934 - val_accuracy: 0.0073
Epoch 5/20
8/8 [==============================] - 46s 6s/step - loss: 4.7776 - accuracy:
0.0117 - val_loss: 4.7912 - val_accuracy: 0.0073
Epoch 6/20
8/8 [==============================] - 49s 7s/step - loss: 4.7909 - accuracy:
0.0124 - val_loss: 4.7881 - val_accuracy: 0.0078
```

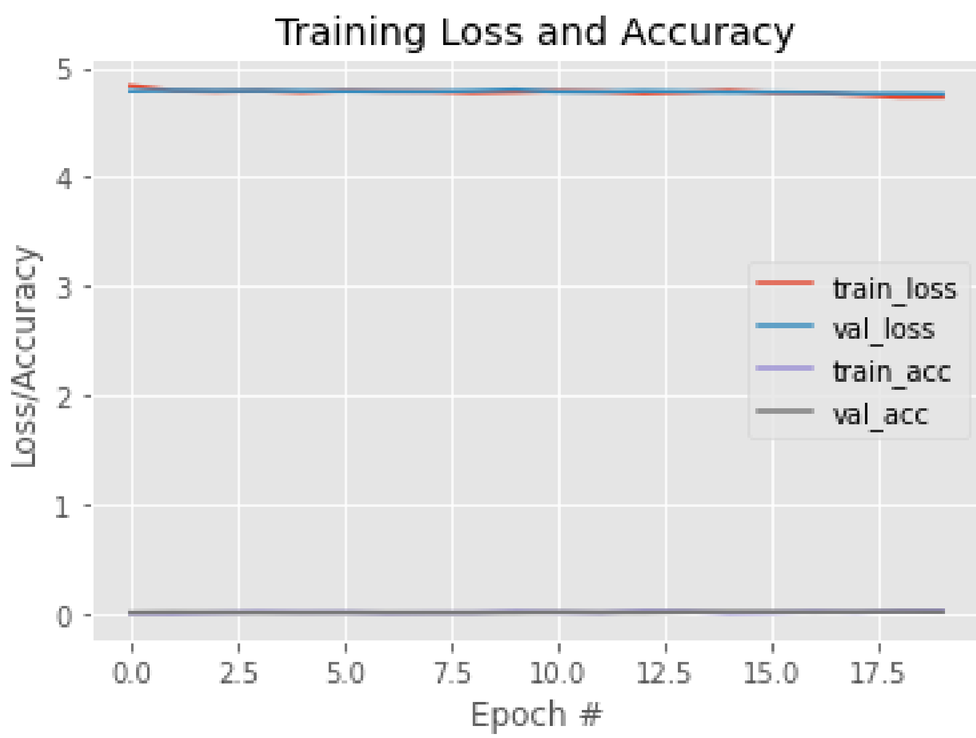
```
Epoch 7/20
8/8 [==============================] - 46s 6s/step - loss: 4.7858 - accuracy:
0.0039 - val_loss: 4.7866 - val_accuracy: 0.0083
Epoch 8/20
8/8 [==============================] - 46s 6s/step - loss: 4.7846 - accuracy:
0.0078 - val_loss: 4.7857 - val_accuracy: 0.0068
Epoch 9/20
8/8 [==============================] - 46s 6s/step - loss: 4.7727 - accuracy:
0.0078 - val_loss: 4.7920 - val_accuracy: 0.0073
Epoch 10/20
8/8 [==============================] - 46s 6s/step - loss: 4.7795 - accuracy:
0.0195 - val_loss: 4.8031 - val_accuracy: 0.0078
Epoch 11/20
8/8 [==============================] - 46s 6s/step - loss: 4.7904 - accuracy:
0.0117 - val_loss: 4.7833 - val_accuracy: 0.0127
Epoch 12/20
8/8 [==============================] - 46s 6s/step - loss: 4.7843 - accuracy:
0.0078 - val_loss: 4.7828 - val_accuracy: 0.0103
Epoch 13/20
8/8 [==============================] - 46s 6s/step - loss: 4.7674 - accuracy:
0.0234 - val_loss: 4.7911 - val_accuracy: 0.0098
Epoch 14/20
8/8 [==============================] - 46s 6s/step - loss: 4.7775 - accuracy:
0.0195 - val_loss: 4.7853 - val_accuracy: 0.0147
Epoch 15/20
8/8 [==============================] - 46s 6s/step - loss: 4.7932 - accuracy:
0.0039 - val_loss: 4.7789 - val_accuracy: 0.0176
Epoch 16/20
8/8 [==============================] - 46s 6s/step - loss: 4.7740 - accuracy:
0.0078 - val_loss: 4.7795 - val_accuracy: 0.0166
Epoch 17/20
8/8 [==============================] - 46s 6s/step - loss: 4.7708 - accuracy:
0.0195 - val_loss: 4.7755 - val_accuracy: 0.0132
Epoch 18/20
8/8 [==============================] - 46s 6s/step - loss: 4.7552 - accuracy:
0.0117 - val_loss: 4.7662 - val_accuracy: 0.0161
Epoch 19/20
8/8 [==============================] - 46s 6s/step - loss: 4.7358 - accuracy:
0.0234 - val_loss: 4.7672 - val_accuracy: 0.0181
Epoch 20/20
8/8 [==============================] - 46s 6s/step - loss: 4.7368 - accuracy:
0.0273 - val_loss: 4.7627 - val_accuracy: 0.0157
```

1.5. MONITORIZACION DEL PROCESO DE ENTRENAMIENTO PARA LA TOMA DE DECISIONES

Entonces vamos a ver la gráfica generada, que como se puede observar en el entrenamiento tuvo una pérdida alta y un accuracy que llegó al 10%, por lo cual seguiremos mejorando para q la pérdida disminuya

[78]:
```python
# Muestro gráfica de accuracy y losses
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 20), H1.history["loss"], label="train_loss")
plt.plot(np.arange(0, 20), H1.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 20), H1.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, 20), H1.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
```

[78]: <matplotlib.legend.Legend at 0x7f400cb45d50>



1.6. EVALUACION DEL MODELO PREDICTIVO Y PLANTEAMIENTO DE LA SIGUIENTE PRUEBA EXPERIMENTAL

[ ]:
```python
from sklearn.metrics import classification_report
import tensorflow as tf

# Se evalúa el modelo de predicción
print("[INFO]: Evaluando red neuronal...")
predictions = model.predict(validation_generator, batch_size=32)
y_preds = tf.argmax(predictions, axis=1)
```

```
y_true = validation_generator.classes

print(classification_report(y_true=y_true,
                            y_pred=y_preds))
```

Como se pudo apreciar en el entrenamiento existe se debe ajustar los parámetros ya que no llegan a converger.

Para ello se agrega BatchNormalization, y nuevas capas de convolución y al final nuevas capas Dense

[14]:
```
# Mejora de la arquitectura, agregando nuevas capas convolutivas

import numpy as np
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Input, Conv2D, Activation, Flatten, Dense,␣
 ↪Dropout, BatchNormalization, MaxPooling2D, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import SGD, Adam
from keras import regularizers

modelo_nuevo = Sequential()

# convolucion 1
modelo_nuevo.add(Conv2D(16, (3,3), input_shape=(224, 224, 3)))
modelo_nuevo.add(BatchNormalization(axis=3))
modelo_nuevo.add(Activation('relu'))
# max pool 1
modelo_nuevo.add(MaxPooling2D(pool_size=(2,2),strides=2))

# convolucion 2
modelo_nuevo.add(Conv2D(32, (3,3)))
modelo_nuevo.add(BatchNormalization(axis=3))
modelo_nuevo.add(Activation('relu'))
# max pool 2
modelo_nuevo.add(MaxPooling2D(pool_size=(2,2),strides=2))

# convolucion 3
modelo_nuevo.add(Conv2D(48, (3,3)))
modelo_nuevo.add(BatchNormalization(axis=3))
modelo_nuevo.add(Activation('relu'))
# max pool 3
modelo_nuevo.add(MaxPooling2D(pool_size=(2,2),strides=2))

# convolucion 4
modelo_nuevo.add(Conv2D(64, (3,3)))
```

```python
modelo_nuevo.add(BatchNormalization(axis=3))
modelo_nuevo.add(Activation('relu'))
# max pool 4
modelo_nuevo.add(MaxPooling2D(pool_size=(2,2),strides=2))

# flatten
modelo_nuevo.add(Flatten())

# dense 1
modelo_nuevo.add(Dense(1024, activation='relu'))

# dense 2
modelo_nuevo.add(Dense(512, activation='relu'))

# dense 3
modelo_nuevo.add(Dense(256, activation='relu'))

# dense 4
modelo_nuevo.add(Dense(120, activation='softmax'))

modelo_nuevo.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])
print(modelo_nuevo.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 222, 222, 16)      448

 batch_normalization (BatchN  (None, 222, 222, 16)     64
 ormalization)

 activation (Activation)     (None, 222, 222, 16)      0

 max_pooling2d_4 (MaxPooling  (None, 111, 111, 16)     0
 2D)

 conv2d_5 (Conv2D)           (None, 109, 109, 32)      4640

 batch_normalization_1 (Batc  (None, 109, 109, 32)     128
 hNormalization)

 activation_1 (Activation)   (None, 109, 109, 32)      0

 max_pooling2d_5 (MaxPooling  (None, 54, 54, 32)       0
 2D)
```

```
 conv2d_6 (Conv2D)           (None, 52, 52, 48)        13872

 batch_normalization_2 (Batc  (None, 52, 52, 48)       192
 hNormalization)

 activation_2 (Activation)   (None, 52, 52, 48)        0

 max_pooling2d_6 (MaxPooling  (None, 26, 26, 48)       0
 2D)

 conv2d_7 (Conv2D)           (None, 24, 24, 64)        27712

 batch_normalization_3 (Batc  (None, 24, 24, 64)       256
 hNormalization)

 activation_3 (Activation)   (None, 24, 24, 64)        0

 max_pooling2d_7 (MaxPooling  (None, 12, 12, 64)       0
 2D)

 flatten_1 (Flatten)         (None, 9216)              0

 dense_2 (Dense)             (None, 1024)              9438208

 dense_3 (Dense)             (None, 512)               524800

 dense_4 (Dense)             (None, 256)               131328

 dense_5 (Dense)             (None, 120)               30840

=================================================================
Total params: 10,172,488
Trainable params: 10,172,168
Non-trainable params: 320
_________________________________________________________________
None
```

En este caso se aumenta el número de pasos por época dependiendo del len de train generator

[19]:
```python
H2=modelo_nuevo.fit(train_generator,
                      steps_per_epoch = 256,
                      validation_data = validation_generator,
                      validation_steps = 64,
                      epochs = 20,
                      verbose = 1)
```

```
Epoch 1/20
```

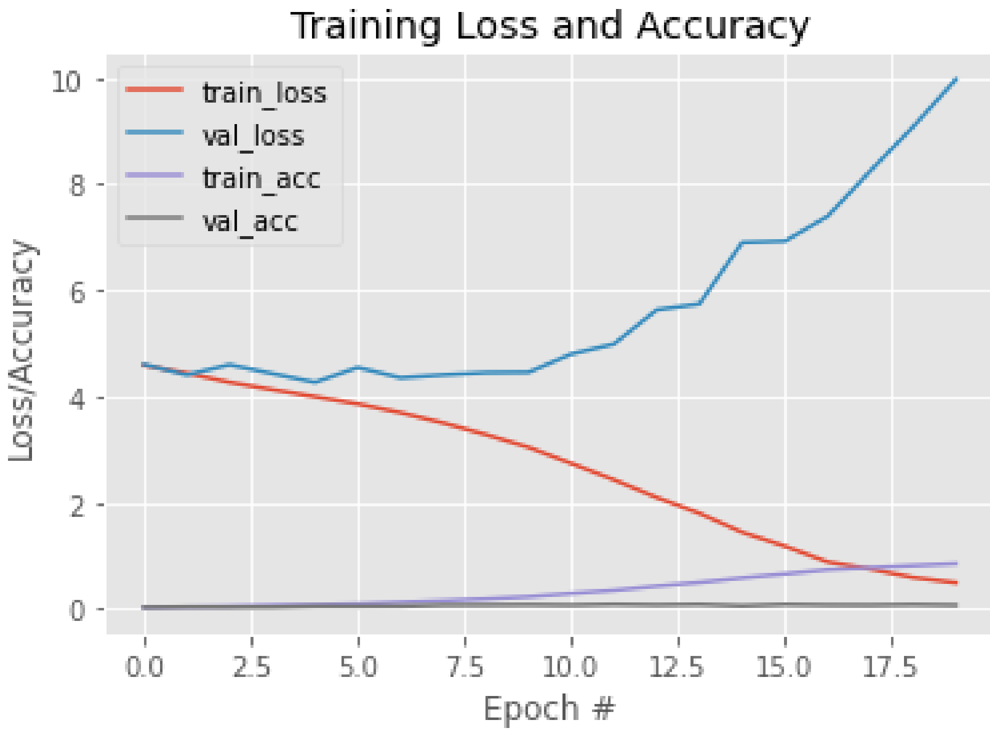
```
256/256 [==============================] - 466s 2s/step - loss: 4.5874 -
accuracy: 0.0251 - val_loss: 4.6045 - val_accuracy: 0.0284
Epoch 2/20
256/256 [==============================] - 461s 2s/step - loss: 4.4409 -
accuracy: 0.0351 - val_loss: 4.4066 - val_accuracy: 0.0362
Epoch 3/20
256/256 [==============================] - 468s 2s/step - loss: 4.2673 -
accuracy: 0.0481 - val_loss: 4.5991 - val_accuracy: 0.0318
Epoch 4/20
256/256 [==============================] - 482s 2s/step - loss: 4.1293 -
accuracy: 0.0670 - val_loss: 4.4321 - val_accuracy: 0.0333
Epoch 5/20
256/256 [==============================] - 480s 2s/step - loss: 3.9958 -
accuracy: 0.0761 - val_loss: 4.2679 - val_accuracy: 0.0499
Epoch 6/20
256/256 [==============================] - 477s 2s/step - loss: 3.8602 -
accuracy: 0.0945 - val_loss: 4.5470 - val_accuracy: 0.0504
Epoch 7/20
256/256 [==============================] - 481s 2s/step - loss: 3.6952 -
accuracy: 0.1245 - val_loss: 4.3532 - val_accuracy: 0.0533
Epoch 8/20
256/256 [==============================] - 480s 2s/step - loss: 3.5009 -
accuracy: 0.1537 - val_loss: 4.4068 - val_accuracy: 0.0734
Epoch 9/20
256/256 [==============================] - 480s 2s/step - loss: 3.2842 -
accuracy: 0.1882 - val_loss: 4.4498 - val_accuracy: 0.0749
Epoch 10/20
256/256 [==============================] - 473s 2s/step - loss: 3.0440 -
accuracy: 0.2289 - val_loss: 4.4559 - val_accuracy: 0.0783
Epoch 11/20
256/256 [==============================] - 471s 2s/step - loss: 2.7403 -
accuracy: 0.2907 - val_loss: 4.8006 - val_accuracy: 0.0724
Epoch 12/20
256/256 [==============================] - 471s 2s/step - loss: 2.4314 -
accuracy: 0.3486 - val_loss: 4.9891 - val_accuracy: 0.0856
Epoch 13/20
256/256 [==============================] - 469s 2s/step - loss: 2.0968 -
accuracy: 0.4290 - val_loss: 5.6347 - val_accuracy: 0.0758
Epoch 14/20
256/256 [==============================] - 469s 2s/step - loss: 1.8002 -
accuracy: 0.4976 - val_loss: 5.7443 - val_accuracy: 0.0841
Epoch 15/20
256/256 [==============================] - 465s 2s/step - loss: 1.4411 -
accuracy: 0.5828 - val_loss: 6.9014 - val_accuracy: 0.0602
Epoch 16/20
256/256 [==============================] - 468s 2s/step - loss: 1.1780 -
accuracy: 0.6592 - val_loss: 6.9259 - val_accuracy: 0.0832
Epoch 17/20
```

```
256/256 [==============================] - 467s 2s/step - loss: 0.8843 -
accuracy: 0.7402 - val_loss: 7.3980 - val_accuracy: 0.0739
Epoch 18/20
256/256 [==============================] - 465s 2s/step - loss: 0.7502 -
accuracy: 0.7760 - val_loss: 8.2515 - val_accuracy: 0.0719
Epoch 19/20
256/256 [==============================] - 466s 2s/step - loss: 0.5914 -
accuracy: 0.8163 - val_loss: 9.0824 - val_accuracy: 0.0807
Epoch 20/20
256/256 [==============================] - 466s 2s/step - loss: 0.4938 -
accuracy: 0.8523 - val_loss: 9.9753 - val_accuracy: 0.0714
```

Se puede observar que con estos parámetros definidos la pérdida ha bajado y el accuracy del entrenamiento va en aumento, mientras que contrariamente ocurre en la data de validation

[20]:
```
# Muestro gráfica de accuracy y losses
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 20), H2.history["loss"], label="train_loss")
plt.plot(np.arange(0, 20), H2.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 20), H2.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, 20), H2.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
```

[20]:
```
<matplotlib.legend.Legend at 0x7f582c498610>
```



[22]:
```
predictions = model.predict(validation_generator, batch_size=32, verbose=1)
print(predictions.shape)
```

```
64/64 [==============================] - 28s 440ms/step
(2044, 120)
```

[21]:
```
from sklearn.metrics import classification_report
import tensorflow as tf

# Evaluando el modelo de predicción con las imágenes de test
print("[INFO]: Evaluando red neuronal...")
predictions = model.predict(validation_generator, batch_size=32)
y_preds = tf.argmax(predictions, axis=1)

y_true = validation_generator.classes

print(classification_report(y_true=y_true,
                            y_pred=y_preds))
```

```
[INFO]: Evaluando red neuronal…
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        19
           1       0.00      0.00      0.00        21
```

2	0.00	0.00	0.00	13
3	0.00	0.00	0.00	23
4	0.00	0.00	0.00	13
5	0.00	0.00	0.00	18
6	0.00	0.00	0.00	21
7	0.00	0.00	0.00	26
8	0.00	0.00	0.00	18
9	0.00	0.00	0.00	17
10	0.00	0.00	0.00	27
11	0.00	0.00	0.00	26
12	0.00	0.00	0.00	13
13	0.00	0.00	0.00	24
14	0.00	0.00	0.00	23
15	0.00	0.00	0.00	15
16	0.00	0.00	0.00	14
17	0.00	0.00	0.00	21
18	0.00	0.00	0.00	15
19	0.00	0.00	0.00	21
20	0.00	0.00	0.00	19
21	0.00	0.00	0.00	15
22	0.00	0.00	0.00	11
23	0.00	0.00	0.00	8
24	0.00	0.00	0.00	14
25	0.00	0.00	0.00	16
26	0.00	0.00	0.00	18
27	0.00	0.00	0.00	18
28	0.00	0.00	0.00	20
29	0.00	0.00	0.00	17
30	0.00	0.00	0.00	23
31	0.00	0.00	0.00	16
32	0.00	0.00	0.00	15
33	0.00	0.00	0.00	20
34	0.00	0.00	0.00	14
35	0.00	0.00	0.00	18
36	0.00	0.00	0.00	20
37	0.00	0.00	0.00	14
38	0.00	0.00	0.00	11
39	1.00	0.05	0.10	20
40	0.00	0.00	0.00	15
41	0.00	0.00	0.00	13
42	0.00	0.00	0.00	18
43	0.00	0.00	0.00	12
44	0.00	0.00	0.00	15
45	0.00	0.00	0.00	12
46	0.00	0.00	0.00	10
47	0.00	0.00	0.00	13
48	0.00	0.00	0.00	21
49	0.00	0.00	0.00	16

50	0.00	0.00	0.00	21
51	0.00	0.00	0.00	10
52	0.00	0.00	0.00	18
53	0.00	0.00	0.00	13
54	0.00	0.00	0.00	16
55	0.00	0.00	0.00	16
56	0.00	0.00	0.00	21
57	0.00	0.00	0.00	16
58	0.00	0.00	0.00	13
59	0.00	0.00	0.00	16
60	0.00	0.00	0.00	19
61	0.00	0.00	0.00	17
62	0.00	0.00	0.00	13
63	0.00	0.00	0.00	17
64	0.01	0.46	0.01	13
65	0.00	0.00	0.00	13
66	0.00	0.00	0.00	13
67	0.02	0.12	0.04	17
68	0.00	0.00	0.00	26
69	0.00	0.00	0.00	17
70	0.01	0.05	0.01	19
71	0.00	0.00	0.00	20
72	0.00	0.00	0.00	13
73	0.00	0.00	0.00	19
74	0.00	0.00	0.00	14
75	0.00	0.00	0.00	18
76	0.00	0.00	0.00	13
77	0.00	0.00	0.00	15
78	0.00	0.00	0.00	15
79	0.00	0.00	0.00	19
80	0.00	0.00	0.00	20
81	0.00	0.00	0.00	15
82	0.00	0.00	0.00	21
83	0.00	0.00	0.00	13
84	0.00	0.00	0.00	19
85	0.00	0.00	0.00	14
86	0.00	0.00	0.00	16
87	0.00	0.13	0.01	15
88	0.00	0.00	0.00	22
89	0.08	0.06	0.07	17
90	0.00	0.00	0.00	19
91	0.00	0.00	0.00	14
92	0.00	0.00	0.00	10
93	0.00	0.00	0.00	25
94	0.00	0.00	0.00	24
95	0.00	0.00	0.00	18
96	0.00	0.00	0.00	17
97	0.00	0.00	0.00	23

98	0.00	0.00	0.00	14
99	0.00	0.00	0.00	16
100	0.00	0.00	0.00	21
101	0.00	0.00	0.00	18
102	0.00	0.00	0.00	23
103	0.00	0.00	0.00	10
104	0.00	0.00	0.00	13
105	0.00	0.00	0.00	18
106	0.00	0.00	0.00	16
107	0.00	0.00	0.00	14
108	0.00	0.00	0.00	15
109	0.00	0.00	0.00	17
110	0.00	0.00	0.00	24
111	0.00	0.00	0.00	16
112	0.00	0.00	0.00	13
113	0.00	0.00	0.00	18
114	0.00	0.00	0.00	15
115	0.00	0.00	0.00	22
116	0.00	0.00	0.00	17
117	0.00	0.00	0.00	22
118	0.00	0.00	0.00	10
119	0.00	0.00	0.00	20
accuracy			0.01	2044
macro avg	0.01	0.01	0.00	2044
weighted avg	0.01	0.01	0.00	2044

Luego de realizar el ajuste en los parámetros se puede observar que todavía falta que haya precisión y mejorar nuestra red, para ello se ha realizado la segunda parte sobre una red pre-entrenada, con mayor número de muestras como se indica a continuación en el punto 2.

2) Estrategia 2: Red pre-entrenada

La segunda estrategia a comparar debe incluir la utilización de una red preentrenada con el dataset ImageNet, llevando a cabo tareas de transfer learning y fine-tuning para resolver la tarea de clasificación asignada. Deben compararse al menos dos tipos de arquitecturas (VGGs, ResNet50, Xception, InceptionV3, InceptionResNetV2, MobileNetV2, DenseNet, ResNet) y se debe seleccionar la que mayor precisión proporcione (información sobre las arquitecturas disponibles en https://keras.io/applications/). Se espera que el/la alumnx utilice todas las técnicas de regularización mostradas en clase de forma justificada para la mejora del rendimiento de la red neuronal (weight regularization, dropout, batch normalization, data augmentation, etc.).

2.1. CONJUNTO DE DATOS

[ ]:
```
import zipfile
def extract_zip_file(file_path):
    with zipfile.ZipFile(file_path, 'r') as zip_ref:
        zip_ref.extractall(".")
```

[ ]:
```
extract_zip_file("./my_dataset/test.zip")
extract_zip_file("./my_dataset/train.zip")
```

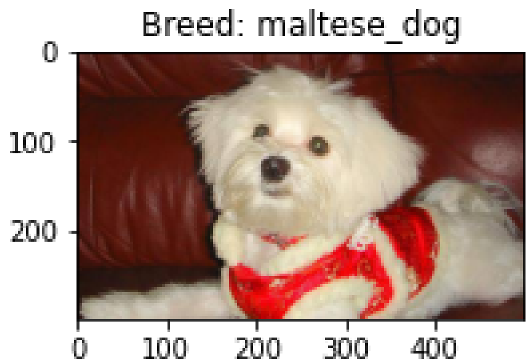
2.2. INSPECCION DE CONJUNTO DE DATOS

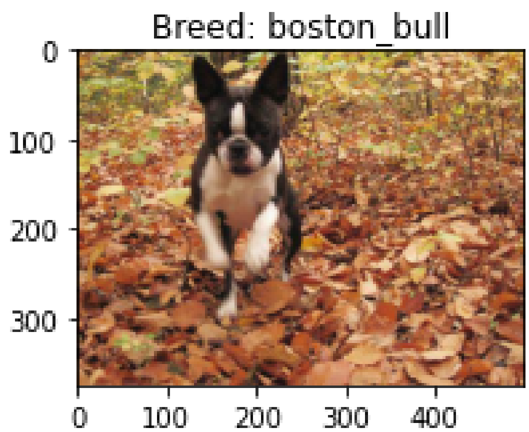
[ ]:
```
import numpy as np
import pandas as pd
import os
import tensorflow as tf
```

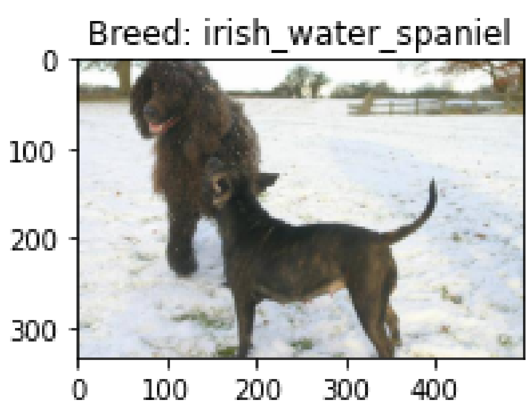
[22]:
```
# Se crea una función para visualizar los ejemplos aleatoriamente

def display_imagen(value):
  # Se selecciona la imagen
  image = cv2.imread("my_dataset/train/" + df.id.values[value])
  # Seleccionar el label
  label = df.breed.values[value]
  # Se muestra la imagen
  fig = plt.figure(figsize=(3,3))
  plt.title('Breed: %s' % label)
  img = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
  plt.imshow(img)

display_imagen(np.random.randint(0, data_batch.shape[0]))
display_imagen(np.random.randint(0, data_batch.shape[0]))
display_imagen(np.random.randint(0, data_batch.shape[0]))
display_imagen(np.random.randint(0, data_batch.shape[0]))
display_imagen(np.random.randint(0, data_batch.shape[0]))
display_imagen(np.random.randint(0, data_batch.shape[0]))
display_imagen(np.random.randint(0, data_batch.shape[0]))
```

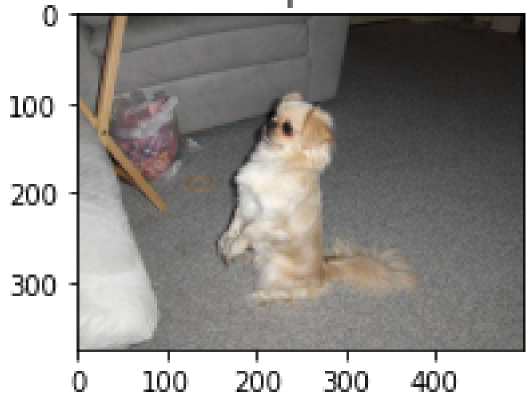

Breed: maltese_dog


Breed: boston_bull

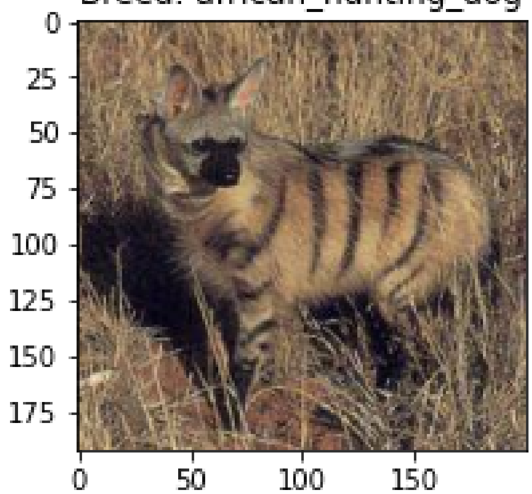

Breed: irish_water_spaniel

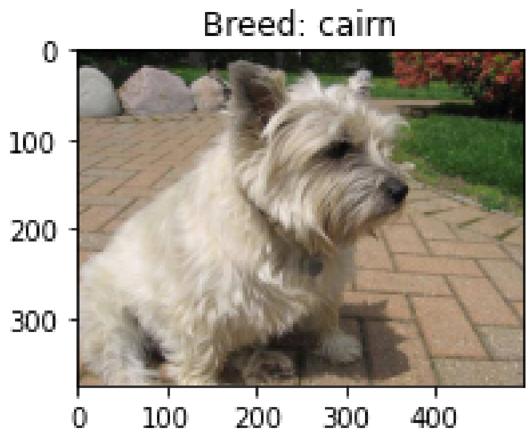
Breed: bedlington_terrier

Breed: pekinese



Breed: african_hunting_dog





2.3. CONSTRUCCION DEL DATAFRAME DE ENTRENAMIENTO

[ ]:
```
def construct_train_df():
    image_list = []
    for root, dirs, filenames in os.walk("./train/"):
        for filename in filenames:
            is_dog = 1 if "dog" in filename else 0
            image_list.append({"file_path": f'./train/{filename}', 'breed':␣
↪breed})
    return pd.DataFrame(image_list)
```

[ ]:
```
train_df = construct_train_df()
print(train_df.shape)
```

```
(25000, 2)
```

2.4. CONSTRUCCION DE DATOS DE PRUEBA Y ENTRENAMIENTO

[ ]:
```
import cv2
x, y = [], []
for index, row in train_df.iterrows():
    image = cv2.imread(row['file_path'])
    image = cv2.resize(image,(64,64))
    image = image / 255
    x.append(image)
    y.append(row['breed'])
```

Transformando a una matriz numpy para que tensorflow pueda trabajar

[ ]:
```
x, y  = np.array(x),np.array(y)
```

[ ]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
↪random_state=1)
```

2.5. CONSTRUCCION DEL MODELO DE DEEP LEARNING

[ ]:
```
import warnings
warnings.filterwarnings('ignore')
pd.set_option("display.max_columns", None)
import keras_tuner as kt
```

[12]:
```
def resumen(model=None):
    '''
    '''
    header = '{:4} {:16} {:24} {:24} {:10}'.format('#',␣
↪'Capa','Input','Output','Parametros'
    )
    print('='*(len(header)))
    print(header)
    print('='*(len(header)))
    count=0
    count_trainable=0
    for i, layer in enumerate(model.layers):
        count_trainable += layer.count_params() if layer.trainable else 0
        input_shape = '{}'.format(layer.input_shape)
        output_shape = '{}'.format(layer.output_shape)
        str = '{:<4d} {:16} {:24} {:24} {:10}'.format(i,layer.name,␣
↪input_shape, output_shape, layer.count_params())
        print(str)
        count += layer.count_params()
    print('_'*(len(header)))
    print('Total Parameters : ', count)
    print('Total Trainable Parameters : ', count_trainable)
    print('Total No-Trainable Parameters : ', count-count_trainable)


vgg16=None
```

[13]:
```
from tensorflow.keras.applications import VGG16

vgg16 = VGG16(weights='imagenet',
                  include_top=True,
                  input_shape=(224, 224, 3))

resumen(vgg16)
```

```
================================================================================
==
```

```
#    Capa              Input                    Output
Parametros
===================================================================================
==
0    input_2           [(None, 224, 224, 3)]    [(None, 224, 224, 3)]
0
1    block1_conv1      (None, 224, 224, 3)      (None, 224, 224, 64)
1792
2    block1_conv2      (None, 224, 224, 64)     (None, 224, 224, 64)
36928
3    block1_pool       (None, 224, 224, 64)     (None, 112, 112, 64)
0
4    block2_conv1      (None, 112, 112, 64)     (None, 112, 112, 128)
73856
5    block2_conv2      (None, 112, 112, 128)    (None, 112, 112, 128)
147584
6    block2_pool       (None, 112, 112, 128)    (None, 56, 56, 128)
0
7    block3_conv1      (None, 56, 56, 128)      (None, 56, 56, 256)
295168
8    block3_conv2      (None, 56, 56, 256)      (None, 56, 56, 256)
590080
9    block3_conv3      (None, 56, 56, 256)      (None, 56, 56, 256)
590080
10   block3_pool       (None, 56, 56, 256)      (None, 28, 28, 256)
0
11   block4_conv1      (None, 28, 28, 256)      (None, 28, 28, 512)
1180160
12   block4_conv2      (None, 28, 28, 512)      (None, 28, 28, 512)
2359808
13   block4_conv3      (None, 28, 28, 512)      (None, 28, 28, 512)
2359808
14   block4_pool       (None, 28, 28, 512)      (None, 14, 14, 512)
0
15   block5_conv1      (None, 14, 14, 512)      (None, 14, 14, 512)
2359808
16   block5_conv2      (None, 14, 14, 512)      (None, 14, 14, 512)
2359808
17   block5_conv3      (None, 14, 14, 512)      (None, 14, 14, 512)
2359808
18   block5_pool       (None, 14, 14, 512)      (None, 7, 7, 512)
0
19   flatten           (None, 7, 7, 512)        (None, 25088)
0
20   fc1               (None, 25088)            (None, 4096)
102764544
21   fc2               (None, 4096)             (None, 4096)
16781312
```

```
22   predictions      (None, 4096)           (None, 1000)
4097000
--------------------------------------------------------------------------------
--
Total Parameters :  138357544
Total Trainable Parameters :  138357544
Total No-Trainable Parameters :  0
```

[ ]:
```python
import os
import numpy as np
from tqdm import tqdm
from keras.applications.imagenet_utils import preprocess_input

from keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(rescale=1./255,
                             preprocessing_function=preprocess_input)
batch_size = 20

def extract_features(directory, sample_count):

    features  = np.zeros(shape=(sample_count, 4, 4, 512))
    labels    = np.zeros(shape=(sample_count))

    generator = datagen.flow_from_directory(directory,
                                            target_size = (150, 150),
                                            batch_size  = batch_size,
                                            class_mode  = 'binary')

    rango = list(range(int(sample_count/batch_size)))
    i = 0
    with tqdm(total=len(rango)) as pbar:
        for inputs_batch, labels_batch in tqdm(generator):

            # características predichas
            features_batch = vgg16.predict(inputs_batch)

            # datos y etiquetas
            features[i * batch_size : (i + 1) * batch_size] = features_batch
            labels  [i * batch_size : (i + 1) * batch_size] = labels_batch
            i += 1
            if i * batch_size >= sample_count:
                break
            pbar.update(1)

    return features, labels
```

```
[ ]: train_features, train_labels           = extract_features(x_train, 2000)
     validation_features, validation_labels = extract_features(y_train, 1000)
     test_features, test_labels             = extract_features(x_test, 1000)
```

```
[ ]: from tensorflow.keras.models import Sequential
     from tensorflow.keras.models import Model
     from tensorflow.keras.layers import Dense
     from tensorflow.keras.layers import Conv2D
     from tensorflow.keras.layers import Dropout
     from tensorflow.keras.layers import Flatten
     from tensorflow.keras.layers import MaxPooling2D

     model = Sequential()
     # CNN
     model.add(Conv2D(input_shape=(64, 64, 3), activation='relu',␣
      ↪kernel_initializer='he_uniform', kernel_size=(6, 6), filters=12))
     model.add(MaxPooling2D(4, 4))
     model.add(Conv2D(filters=10, kernel_size=(3,3), activation='relu',␣
      ↪kernel_initializer='he_uniform'))
     model.add(MaxPooling2D(2,2))
     # ANN
     model.add(Flatten())
     model.add(Dense(12, activation='relu', kernel_initializer='he_uniform'))
     model.add(Dense(1, activation='sigmoid', kernel_initializer='glorot_uniform'))

     model.compile(loss='binary_crossentropy', optimizer='adam',␣
      ↪metrics=['accuracy'])
     model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 59, 59, 12)        1308

 max_pooling2d (MaxPooling2D  (None, 14, 14, 12)       0
 )

 conv2d_1 (Conv2D)           (None, 12, 12, 10)        1090

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 10)         0
 2D)

 flatten (Flatten)           (None, 360)               0

 dense (Dense)               (None, 12)                4332
```

```
 dense_1 (Dense)             (None, 1)                 13

=================================================================
Total params: 6,743
Trainable params: 6,743
Non-trainable params: 0
_________________________________________________________________
```

User settings:

KMP_AFFINITY=granularity=fine,noverbose,compact,1,0

KMP_BLOCKTIME=0

KMP_DUPLICATE_LIB_OK=True

KMP_INIT_AT_FORK=FALSE

KMP_SETTINGS=1

KMP_WARNINGS=0

Effective settings:

KMP_ABORT_DELAY=0

KMP_ADAPTIVE_LOCK_PROPS='1,1024'

KMP_ALIGN_ALLOC=64

KMP_ALL_THREADPRIVATE=128

KMP_ATOMIC_MODE=2

KMP_BLOCKTIME=0

KMP_CPUINFO_FILE: value is not defined

KMP_DETERMINISTIC_REDUCTION=false

KMP_DEVICE_THREAD_LIMIT=2147483647

KMP_DISP_NUM_BUFFERS=7

KMP_DUPLICATE_LIB_OK=true

KMP_ENABLE_TASK_THROTTLING=true

KMP_FORCE_REDUCTION: value is not defined

KMP_FOREIGN_THREADS_THREADPRIVATE=true

KMP_FORKJOIN_BARRIER='2,2'

KMP_FORKJOIN_BARRIER_PATTERN='hyper,hyper'

KMP_GTID_MODE=3

KMP_HANDLE_SIGNALS=false

KMP_HOT_TEAMS_MAX_LEVEL=1

KMP_HOT_TEAMS_MODE=0

KMP_INIT_AT_FORK=true

KMP_LIBRARY=throughput

KMP_LOCK_KIND=queuing

KMP_MALLOC_POOL_INCR=1M

KMP_NUM_LOCKS_IN_BLOCK=1

KMP_PLAIN_BARRIER='2,2'

KMP_PLAIN_BARRIER_PATTERN='hyper,hyper'

KMP_REDUCTION_BARRIER='1,1'

KMP_REDUCTION_BARRIER_PATTERN='hyper,hyper'

KMP_SCHEDULE='static,balanced;guided,iterative'

KMP_SETTINGS=true

KMP_SPIN_BACKOFF_PARAMS='4096,100'

KMP_STACKOFFSET=64

KMP_STACKPAD=0

KMP_STACKSIZE=8M

KMP_STORAGE_MAP=false

KMP_TASKING=2

KMP_TASKLOOP_MIN_TASKS=0

KMP_TASK_STEALING_CONSTRAINT=1

KMP_TEAMS_THREAD_LIMIT=4

KMP_TOPOLOGY_METHOD=all

KMP_USE_YIELD=1

KMP_VERSION=false

KMP_WARNINGS=false

OMP_AFFINITY_FORMAT='OMP: pid %P tid %i thread %n bound to OS proc set {%A}'

OMP_ALLOCATOR=omp_default_mem_alloc

OMP_CANCELLATION=false

OMP_DEFAULT_DEVICE=0

OMP_DISPLAY_AFFINITY=false

OMP_DISPLAY_ENV=false

OMP_DYNAMIC=false

OMP_MAX_ACTIVE_LEVELS=1

OMP_MAX_TASK_PRIORITY=0

OMP_NESTED: deprecated; max-active-levels-var=1

OMP_NUM_THREADS: value is not defined

OMP_PLACES: value is not defined

OMP_PROC_BIND='intel'

OMP_SCHEDULE='static'

OMP_STACKSIZE=8M

OMP_TARGET_OFFLOAD=DEFAULT

OMP_THREAD_LIMIT=2147483647

OMP_WAIT_POLICY=PASSIVE

KMP_AFFINITY='noverbose,warnings,respect,granularity=fine,compact,1,0'

Ajuste de hiperparámetros

[ ]:
```
import ResNet50V2, preprocess_input

def build_model(hp):

  # Se añade la primera capa densa
  resnet = ResNet50V2(input_shape = [64,64,3], weights='imagenet',␣
↪include_top=False)

  for layer in resnet.layers:
      layer.trainable = False

  # Se agrega batch normalization y pooling
  x = resnet.output
  x = BatchNormalization()(x)
  x = GlobalAveragePooling2D()(x)
  x = Dropout(0.5)(x)

  x = Dense(120, activation='relu')(x)
  x = Dropout(0.5)(x)
  outputs = Dense(units=1, activation='sigmoid')(x)

  model = Model(inputs=inputs, outputs=outputs)
  model.compile(loss='categorical_crossentropy', metrics=['accuracy'],␣
↪optimizer='adam')
  return model
```

[ ]:
```
tuner = kt.RandomSearch(build_model, max_trials=3, overwrite=True,␣
↪objective='val_accuracy', directory="./tuning")
```

[ ]:
```
tuner.search(x_train, y_train, validation_split=0.2, epochs=5, callbacks=[tf.
↪keras.callbacks.TensorBoard("./tensorboard")])
```

```
Trial 3 Complete [00h 12m 23s]
val_accuracy: 0.7772499918937683

Best val_accuracy So Far: 0.7795000076293945
Total elapsed time: 00h 38m 54s
INFO:tensorflow:Oracle triggered exit
```

[ ]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 59, 59, 12)        1308

 max_pooling2d (MaxPooling2D  (None, 14, 14, 12)       0
 )

 conv2d_1 (Conv2D)           (None, 12, 12, 10)        1090

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 10)         0
 2D)

 flatten (Flatten)           (None, 360)               0

 dense (Dense)               (None, 12)                4332

 dense_1 (Dense)             (None, 1)                 13

=================================================================
Total params: 6,743
Trainable params: 6,743
Non-trainable params: 0
_________________________________________________________________
```

2.6. AJUSTE DEL MODELO

[ ]:
```
history = model.fit(x_train, y_train, validation_data=(x_test, y_test),␣
↪epochs=1)
```

```
625/625 [==============================] - 65s 103ms/step - loss: 0.5533 -
accuracy: 0.7151 - val_loss: 0.5262 - val_accuracy: 0.7376
```

Construcción de marco de datos de prueba

```
[ ]: def construct_test_df():
         x = []
         for dirname, _, filenames in os.walk("./test"):
             for filename in filenames:
                 x.append(f'./test/{filename}')
         return pd.DataFrame({'file_path': x})
```

```
[ ]: test_df = construct_test_df()
```

```
[ ]: test_images = []
     for index, row in test_df.iterrows():
         image = cv2.imread(row['file_path'])
         image = cv2.resize(image, (64, 64))
         image = image / 255
         test_images.append(image)
```

```
[ ]: test_images = np.array(test_images)
```

Predicción de los datos de prueba

```
[ ]: y_pred = model.predict(test_images)
```

```
[ ]: y_pred.shape
```

```
[ ]: (12500, 1)
```

```
[ ]: dog = y_pred.reshape(-1)
```

Crear archivo de envío

```
[ ]: submission_df = pd.DataFrame({'id':np.arange(1, len(dog)+1), 'label': (dog > 0.
     ↪5).astype('int')})
```

```
[ ]: submission_df.to_csv("./submission.csv", index=False) #Archivo ejecutado con␣
     ↪los resultados
```

3) Conclusiones

- Se ha desarrollado un modelo que predice la raza a partir de las imágenes de perros en la segunda parte se ha usado la red preentrenada ResNet50V2. Con la cual se obtuvo una precisión del 71,50%, lo cual es bueno, ya que solo hemos todas las 120 razas de perros para el conjunto de entrenamiento y prueba y solo 20 épocas.
- Al entrenar la red neuronal de manera manual se ha tenido una precisión más alta debido a que realizó varios entrenamientos y muchos ajustes de parámetros con varias redes convolucionales
- Otro punto a destacar es que con una red preentrenada se necesita gran cantidad de almacenamiento y más gpu, por esa razón si se agregaría más épocas se podría haber mejorado la

precisión y disminuido más la pérdida