main

December 12, 2022

[1]:
```python
import tensorflow as tf
import tensorflow_datasets as tfds
#Cargar datos de archivo "kagglecatsanddogs_5340.zip"
datos, metadatos = tfds.load('cats_vs_dogs', as_supervised=True, with_info=True)
```

[2]:
```python
#Imprimir los metadatos para revisarlos
metadatos
```

[2]:
```
tfds.core.DatasetInfo(
    name='cats_vs_dogs',
    full_name='cats_vs_dogs/4.0.0',
    description="""
    A large set of images of cats and dogs. There are 1738 corrupted images that
are dropped.
    """,
    homepage='https://www.microsoft.com/en-us/download/details.aspx?id=54765',
    data_path='C:\\Users\\79449\\tensorflow_datasets\\cats_vs_dogs\\4.0.0',
    file_format=tfrecord,
    download_size=786.67 MiB,
    dataset_size=689.64 MiB,
    features=FeaturesDict({
        'image': Image(shape=(None, None, 3), dtype=tf.uint8),
        'image/filename': Text(shape=(), dtype=tf.string),
        'label': ClassLabel(shape=(), dtype=tf.int64, num_classes=2),
    }),
    supervised_keys=('image', 'label'),
    disable_shuffling=False,
    splits={
        'train': <SplitInfo num_examples=23262, num_shards=8>,
    },
    citation="""@Inproceedings (Conference){asirra-a-captcha-that-exploits-
interest-aligned-manual-image-categorization,
    author = {Elson, Jeremy and Douceur, John (JD) and Howell, Jon and Saul,
Jared},
    title = {Asirra: A CAPTCHA that Exploits Interest-Aligned Manual Image
Categorization},
    booktitle = {Proceedings of 14th ACM Conference on Computer and
Communications Security (CCS)},
    year = {2007},
    month = {October},
    publisher = {Association for Computing Machinery, Inc.},
    url = {https://www.microsoft.com/en-us/research/publication/asirra-a-
captcha-that-exploits-interest-aligned-manual-image-categorization/},
    edition = {Proceedings of 14th ACM Conference on Computer and Communications
Security (CCS)},
    }""",
)
```

[5]:
```
#Mostrar un ejemplo de los datos de entremiento del dataset
tfds.as_dataframe(datos['train'].take(10), metadatos)
```

[5]:
```
                                              image  label
0  [[[242, 248, 248], [240, 246, 246], [235, 239,…      1
1  [[[215, 165, 114], [187, 135, 85], [232, 176, …      1
2  [[[177, 183, 157], [185, 191, 165], [192, 198,…      1
3  [[[92, 66, 7], [93, 67, 8], [93, 67, 8], [93, …      0
4  [[[140, 138, 141], [140, 138, 141], [141, 139,…      1
5  [[[126, 128, 125], [114, 116, 111], [97, 98, 9…      1
6  [[[40, 46, 70], [33, 38, 57], [30, 34, 59], [3…      0
7  [[[17, 11, 25], [26, 20, 34], [43, 36, 52], [6…      0
8  [[[81, 78, 71], [65, 62, 55], [49, 46, 39], [4…      1
9  [[[40, 40, 40], [40, 40, 40], [40, 40, 40], [4…      1
```

[6]:
```
#Lo pasamos a TAMANO_IMG (100x100) y lo convertimos a blanco y negro *(solo␣
 ↪para visualizar)*
import matplotlib.pyplot as plt
import cv2

plt.figure(figsize=(20,20))

TAMANO_IMG=100

for i, (imagen, etiqueta) in enumerate(datos['train'].take(25)):
  imagen = cv2.resize(imagen.numpy(), (TAMANO_IMG, TAMANO_IMG))
  imagen = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)
  plt.subplot(5, 5, i+1)
  plt.xticks([])
  plt.yticks([])
  plt.imshow(imagen, cmap='gray')
```

[7]:
```
#Creamos la variable que contendra todos los pares de los datos (imagen y␣
↪etiqueta) ya modificados (blanco y negro, 100x100)
datos_entrenamiento = []
```

[8]:
```
#Guardamos las imagenes en un tamaño de TAMANO_IMG (100x100) y lo convertimos a␣
↪blanco y negro
for i, (imagen, etiqueta) in enumerate(datos['train']): #Todos los datos
  imagen = cv2.resize(imagen.numpy(), (TAMANO_IMG, TAMANO_IMG))
  imagen = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)
  imagen = imagen.reshape(TAMANO_IMG, TAMANO_IMG, 1) #Cambiar tamano a 100,100,1
  datos_entrenamiento.append([imagen, etiqueta]) #guardamos en la nueva␣
↪variable creada anteriormente.
```

[10]:
```
#Ver cuantos datos tengo en la variable
len(datos_entrenamiento)
```

[10]:
```
23262
```

[11]:
```
#separamos datos
#Preparar mis variables X (entradas) y y (etiquetas) separadas
X = [] #imagenes de entrada (pixeles / fotos)
y = [] #etiquetas (perro o gato / indice)

for imagen, etiqueta in datos_entrenamiento:
  X.append(imagen)
  y.append(etiqueta)
```

[13]:
```
#Normalizar los datos de las X (imagenes). Se pasan a numero flotante y dividen␣
 ↪entre 255 para quedar de 0-1 en lugar de 0-255
#Para poder entrenar de manera optima
import numpy as np

X = np.array(X).astype(float) / 255
```

[16]:
```
#Convertir etiquetas en arreglo simple
y = np.array(y)
```

[17]:
```
#Datos tipo array de numpy
y
```

[17]:
```
array([1, 1, 1, …, 0, 1, 0], dtype=int64)
```

[18]:
```
#Los tamaños de "X" y "y"
print(X.shape)
print(y.shape)
```

```
(23262, 100, 100, 1)
(23262,)
```

[19]:
```
#Crear los modelos iniciales
#Usan sigmoid como salida (en lugar de softmax) para mostrar como podria␣
 ↪funcionar con dicha funcion de activacion.
#Sigmoid regresa siempre datos entre 0 y 1. Realizamos el entrenamiento para al␣
 ↪final considerar que si la respuesta se
#acerca a 0, es un gato, y si se acerca a 1, es un perro.

modeloDenso = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(100, 100, 1)),
  tf.keras.layers.Dense(150, activation='relu'),
  tf.keras.layers.Dense(150, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])

modeloCNN = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(100, 100,␣
 ↪1)),
  tf.keras.layers.MaxPooling2D(2, 2),
  tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2, 2),
  tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2, 2),

  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(100, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])

modeloCNN2 = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(100, 100,␣
 ↪1)),
  tf.keras.layers.MaxPooling2D(2, 2),
  tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2, 2),
  tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2, 2),

  tf.keras.layers.Dropout(0.5),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(250, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])
```

[20]:
```
#Compilar modelos. Usar crossentropy binario ya que tenemos solo 2 opciones␣
 ↪(perro o gato)
modeloDenso.compile(optimizer='adam',
                    loss='binary_crossentropy',
                    metrics=['accuracy'])

modeloCNN.compile(optimizer='adam',
                    loss='binary_crossentropy',
                    metrics=['accuracy'])

modeloCNN2.compile(optimizer='adam',
                    loss='binary_crossentropy',
                    metrics=['accuracy'])
```

[21]:
```
from tensorflow.keras.callbacks import TensorBoard
```

```
[22]: #La variable de tensorboard se envia en el arreglo de "callbacks" (hay otros␣
       ↪tipos de callbacks soportados)
      #En este caso guarda datos en la carpeta indicada en cada epoca, de manera que␣
       ↪despues
      #Tensorboard los lee para hacer graficas
      tensorboardDenso = TensorBoard(log_dir='logs/denso')
      modeloDenso.fit(X, y, batch_size=32,
                      validation_split=0.15,
                      epochs=30,
                      callbacks=[tensorboardDenso])
```

```
Epoch 1/30
618/618 [==============================] - 9s 13ms/step - loss: 0.7133 -
accuracy: 0.5408 - val_loss: 0.6738 - val_accuracy: 0.5825
Epoch 2/30
618/618 [==============================] - 8s 12ms/step - loss: 0.6755 -
accuracy: 0.5756 - val_loss: 0.6757 - val_accuracy: 0.5897
Epoch 3/30
618/618 [==============================] - 8s 13ms/step - loss: 0.6667 -
accuracy: 0.5939 - val_loss: 0.6708 - val_accuracy: 0.5914
Epoch 4/30
618/618 [==============================] - 8s 12ms/step - loss: 0.6702 -
accuracy: 0.5820 - val_loss: 0.6701 - val_accuracy: 0.5940
Epoch 5/30
618/618 [==============================] - 8s 13ms/step - loss: 0.6630 -
accuracy: 0.5971 - val_loss: 0.6686 - val_accuracy: 0.5968
Epoch 6/30
618/618 [==============================] - 8s 12ms/step - loss: 0.6637 -
accuracy: 0.6001 - val_loss: 0.6940 - val_accuracy: 0.5562
Epoch 7/30
618/618 [==============================] - 8s 12ms/step - loss: 0.6598 -
accuracy: 0.6039 - val_loss: 0.6662 - val_accuracy: 0.5954
Epoch 8/30
618/618 [==============================] - 8s 12ms/step - loss: 0.6609 -
accuracy: 0.6012 - val_loss: 0.6693 - val_accuracy: 0.5854
Epoch 9/30
618/618 [==============================] - 8s 13ms/step - loss: 0.6577 -
accuracy: 0.6104 - val_loss: 0.6773 - val_accuracy: 0.5931
Epoch 10/30
618/618 [==============================] - 8s 12ms/step - loss: 0.6570 -
accuracy: 0.6107 - val_loss: 0.6749 - val_accuracy: 0.5797
Epoch 11/30
618/618 [==============================] - 8s 13ms/step - loss: 0.6517 -
accuracy: 0.6180 - val_loss: 0.6669 - val_accuracy: 0.5980
Epoch 12/30
618/618 [==============================] - 8s 12ms/step - loss: 0.6560 -
accuracy: 0.6112 - val_loss: 0.6689 - val_accuracy: 0.5963
```

```
Epoch 13/30
618/618 [==============================] - 8s 13ms/step - loss: 0.6521 -
accuracy: 0.6169 - val_loss: 0.6675 - val_accuracy: 0.5903
Epoch 14/30
618/618 [==============================] - 8s 13ms/step - loss: 0.6514 -
accuracy: 0.6193 - val_loss: 0.6675 - val_accuracy: 0.5928
Epoch 15/30
618/618 [==============================] - 8s 13ms/step - loss: 0.6514 -
accuracy: 0.6163 - val_loss: 0.6643 - val_accuracy: 0.5974
Epoch 16/30
618/618 [==============================] - 8s 12ms/step - loss: 0.6504 -
accuracy: 0.6191 - val_loss: 0.6656 - val_accuracy: 0.6006
Epoch 17/30
618/618 [==============================] - 8s 13ms/step - loss: 0.6482 -
accuracy: 0.6238 - val_loss: 0.6769 - val_accuracy: 0.5828
Epoch 18/30
618/618 [==============================] - 8s 13ms/step - loss: 0.6467 -
accuracy: 0.6260 - val_loss: 0.6711 - val_accuracy: 0.5894
Epoch 19/30
618/618 [==============================] - 8s 13ms/step - loss: 0.6471 -
accuracy: 0.6234 - val_loss: 0.6663 - val_accuracy: 0.5931
Epoch 20/30
618/618 [==============================] - 8s 13ms/step - loss: 0.6495 -
accuracy: 0.6213 - val_loss: 0.6708 - val_accuracy: 0.5911
Epoch 21/30
618/618 [==============================] - 8s 13ms/step - loss: 0.6451 -
accuracy: 0.6270 - val_loss: 0.6748 - val_accuracy: 0.5960
Epoch 22/30
618/618 [==============================] - 7s 12ms/step - loss: 0.6465 -
accuracy: 0.6247 - val_loss: 0.6721 - val_accuracy: 0.6017
Epoch 23/30
618/618 [==============================] - 8s 12ms/step - loss: 0.6454 -
accuracy: 0.6264 - val_loss: 0.6884 - val_accuracy: 0.5825
Epoch 24/30
618/618 [==============================] - 8s 12ms/step - loss: 0.6431 -
accuracy: 0.6307 - val_loss: 0.6693 - val_accuracy: 0.5926
Epoch 25/30
618/618 [==============================] - 8s 12ms/step - loss: 0.6442 -
accuracy: 0.6276 - val_loss: 0.6706 - val_accuracy: 0.5840
Epoch 26/30
618/618 [==============================] - 8s 12ms/step - loss: 0.6413 -
accuracy: 0.6333 - val_loss: 0.6746 - val_accuracy: 0.5940
Epoch 27/30
618/618 [==============================] - 8s 13ms/step - loss: 0.6417 -
accuracy: 0.6331 - val_loss: 0.6809 - val_accuracy: 0.5900
Epoch 28/30
618/618 [==============================] - 8s 12ms/step - loss: 0.6428 -
accuracy: 0.6296 - val_loss: 0.6714 - val_accuracy: 0.5920
```

```
Epoch 29/30
618/618 [==============================] - 8s 12ms/step - loss: 0.6403 -
accuracy: 0.6349 - val_loss: 0.6716 - val_accuracy: 0.5954
Epoch 30/30
618/618 [==============================] - 7s 12ms/step - loss: 0.6411 -
accuracy: 0.6316 - val_loss: 0.6755 - val_accuracy: 0.5946
```

[22]:
```
<keras.callbacks.History at 0x2975aef7d90>
```

[23]:
```
tensorboardCNN = TensorBoard(log_dir='logs/cnn')
modeloCNN.fit(X, y, batch_size=32,
                validation_split=0.15,
                epochs=30,
                callbacks=[tensorboardCNN])
```

```
Epoch 1/30
618/618 [==============================] - 78s 125ms/step - loss: 0.6314 -
accuracy: 0.6308 - val_loss: 0.5563 - val_accuracy: 0.7140
Epoch 2/30
618/618 [==============================] - 76s 122ms/step - loss: 0.4965 -
accuracy: 0.7598 - val_loss: 0.4585 - val_accuracy: 0.7802
Epoch 3/30
618/618 [==============================] - 76s 123ms/step - loss: 0.4262 -
accuracy: 0.8010 - val_loss: 0.4391 - val_accuracy: 0.7883
Epoch 4/30
618/618 [==============================] - 78s 126ms/step - loss: 0.3765 -
accuracy: 0.8275 - val_loss: 0.4107 - val_accuracy: 0.8189
Epoch 5/30
618/618 [==============================] - 77s 124ms/step - loss: 0.3197 -
accuracy: 0.8602 - val_loss: 0.4030 - val_accuracy: 0.8203
Epoch 6/30
618/618 [==============================] - 77s 125ms/step - loss: 0.2714 -
accuracy: 0.8845 - val_loss: 0.3995 - val_accuracy: 0.8312
Epoch 7/30
618/618 [==============================] - 76s 123ms/step - loss: 0.2201 -
accuracy: 0.9068 - val_loss: 0.4259 - val_accuracy: 0.8226
Epoch 8/30
618/618 [==============================] - 77s 125ms/step - loss: 0.1679 -
accuracy: 0.9317 - val_loss: 0.4432 - val_accuracy: 0.8364
Epoch 9/30
618/618 [==============================] - 78s 126ms/step - loss: 0.1191 -
accuracy: 0.9532 - val_loss: 0.5787 - val_accuracy: 0.8206
Epoch 10/30
618/618 [==============================] - 79s 127ms/step - loss: 0.0864 -
accuracy: 0.9676 - val_loss: 0.5446 - val_accuracy: 0.8281
Epoch 11/30
618/618 [==============================] - 80s 130ms/step - loss: 0.0632 -
accuracy: 0.9769 - val_loss: 0.6150 - val_accuracy: 0.8304
```

```
Epoch 12/30
618/618 [==============================] - 81s 131ms/step - loss: 0.0407 -
accuracy: 0.9870 - val_loss: 0.6901 - val_accuracy: 0.8272
Epoch 13/30
618/618 [==============================] - 81s 132ms/step - loss: 0.0407 -
accuracy: 0.9868 - val_loss: 0.7249 - val_accuracy: 0.8264
Epoch 14/30
618/618 [==============================] - 81s 132ms/step - loss: 0.0350 -
accuracy: 0.9888 - val_loss: 0.8723 - val_accuracy: 0.8315
Epoch 15/30
618/618 [==============================] - 81s 130ms/step - loss: 0.0223 -
accuracy: 0.9936 - val_loss: 0.9074 - val_accuracy: 0.8209
Epoch 16/30
618/618 [==============================] - 77s 124ms/step - loss: 0.0284 -
accuracy: 0.9908 - val_loss: 0.8563 - val_accuracy: 0.8166
Epoch 17/30
618/618 [==============================] - 77s 125ms/step - loss: 0.0284 -
accuracy: 0.9899 - val_loss: 0.9684 - val_accuracy: 0.8140
Epoch 18/30
618/618 [==============================] - 77s 125ms/step - loss: 0.0195 -
accuracy: 0.9933 - val_loss: 0.9040 - val_accuracy: 0.8284
Epoch 19/30
618/618 [==============================] - 77s 124ms/step - loss: 0.0316 -
accuracy: 0.9897 - val_loss: 0.9009 - val_accuracy: 0.8183
Epoch 20/30
618/618 [==============================] - 77s 124ms/step - loss: 0.0233 -
accuracy: 0.9916 - val_loss: 0.9898 - val_accuracy: 0.8252
Epoch 21/30
618/618 [==============================] - 77s 125ms/step - loss: 0.0197 -
accuracy: 0.9936 - val_loss: 1.0691 - val_accuracy: 0.8321
Epoch 22/30
618/618 [==============================] - 77s 125ms/step - loss: 0.0197 -
accuracy: 0.9939 - val_loss: 1.0543 - val_accuracy: 0.8289
Epoch 23/30
618/618 [==============================] - 77s 125ms/step - loss: 0.0167 -
accuracy: 0.9942 - val_loss: 1.1542 - val_accuracy: 0.8295
Epoch 24/30
618/618 [==============================] - 78s 126ms/step - loss: 0.0225 -
accuracy: 0.9919 - val_loss: 0.9817 - val_accuracy: 0.8246
Epoch 25/30
618/618 [==============================] - 78s 126ms/step - loss: 0.0215 -
accuracy: 0.9928 - val_loss: 1.1036 - val_accuracy: 0.8135
Epoch 26/30
618/618 [==============================] - 78s 126ms/step - loss: 0.0170 -
accuracy: 0.9945 - val_loss: 1.0926 - val_accuracy: 0.8143
Epoch 27/30
618/618 [==============================] - 78s 126ms/step - loss: 0.0204 -
accuracy: 0.9936 - val_loss: 1.2548 - val_accuracy: 0.8229
```

```
Epoch 28/30
618/618 [==============================] - 78s 126ms/step - loss: 0.0159 -
accuracy: 0.9948 - val_loss: 1.2669 - val_accuracy: 0.8183
Epoch 29/30
618/618 [==============================] - 78s 127ms/step - loss: 0.0153 -
accuracy: 0.9952 - val_loss: 1.3214 - val_accuracy: 0.8066
Epoch 30/30
618/618 [==============================] - 78s 126ms/step - loss: 0.0126 -
accuracy: 0.9957 - val_loss: 1.1977 - val_accuracy: 0.8209
```

```
[23]: <keras.callbacks.History at 0x2975b5055e0>
```

```
[24]: tensorboardCNN2 = TensorBoard(log_dir='logs/cnn2')
      modeloCNN2.fit(X, y, batch_size=32,
                      validation_split=0.15,
                      epochs=30,
                      callbacks=[tensorboardCNN2])
```

```
Epoch 1/30
618/618 [==============================] - 85s 136ms/step - loss: 0.6528 -
accuracy: 0.5984 - val_loss: 0.6007 - val_accuracy: 0.6802
Epoch 2/30
618/618 [==============================] - 85s 138ms/step - loss: 0.5314 -
accuracy: 0.7332 - val_loss: 0.4944 - val_accuracy: 0.7582
Epoch 3/30
618/618 [==============================] - 85s 138ms/step - loss: 0.4653 -
accuracy: 0.7797 - val_loss: 0.4367 - val_accuracy: 0.7974
Epoch 4/30
618/618 [==============================] - 86s 140ms/step - loss: 0.4284 -
accuracy: 0.7990 - val_loss: 0.4026 - val_accuracy: 0.8143
Epoch 5/30
618/618 [==============================] - 86s 138ms/step - loss: 0.3892 -
accuracy: 0.8240 - val_loss: 0.3802 - val_accuracy: 0.8298
Epoch 6/30
618/618 [==============================] - 86s 139ms/step - loss: 0.3545 -
accuracy: 0.8413 - val_loss: 0.3875 - val_accuracy: 0.8298
Epoch 7/30
618/618 [==============================] - 86s 139ms/step - loss: 0.3219 -
accuracy: 0.8571 - val_loss: 0.3778 - val_accuracy: 0.8358
Epoch 8/30
618/618 [==============================] - 86s 139ms/step - loss: 0.2916 -
accuracy: 0.8742 - val_loss: 0.3629 - val_accuracy: 0.8413
Epoch 9/30
618/618 [==============================] - 86s 139ms/step - loss: 0.2591 -
accuracy: 0.8859 - val_loss: 0.3677 - val_accuracy: 0.8441
Epoch 10/30
618/618 [==============================] - 87s 140ms/step - loss: 0.2322 -
accuracy: 0.9032 - val_loss: 0.3697 - val_accuracy: 0.8521
```

```
Epoch 11/30
618/618 [==============================] - 85s 138ms/step - loss: 0.1929 -
accuracy: 0.9216 - val_loss: 0.3710 - val_accuracy: 0.8501
Epoch 12/30
618/618 [==============================] - 84s 136ms/step - loss: 0.1685 -
accuracy: 0.9336 - val_loss: 0.3749 - val_accuracy: 0.8516
Epoch 13/30
618/618 [==============================] - 85s 137ms/step - loss: 0.1529 -
accuracy: 0.9402 - val_loss: 0.3953 - val_accuracy: 0.8530
Epoch 14/30
618/618 [==============================] - 85s 137ms/step - loss: 0.1310 -
accuracy: 0.9492 - val_loss: 0.4019 - val_accuracy: 0.8564
Epoch 15/30
618/618 [==============================] - 85s 137ms/step - loss: 0.1174 -
accuracy: 0.9535 - val_loss: 0.4366 - val_accuracy: 0.8562
Epoch 16/30
618/618 [==============================] - 85s 138ms/step - loss: 0.0999 -
accuracy: 0.9625 - val_loss: 0.4281 - val_accuracy: 0.8530
Epoch 17/30
618/618 [==============================] - 84s 136ms/step - loss: 0.0902 -
accuracy: 0.9668 - val_loss: 0.4443 - val_accuracy: 0.8510
Epoch 18/30
618/618 [==============================] - 84s 136ms/step - loss: 0.0828 -
accuracy: 0.9695 - val_loss: 0.5086 - val_accuracy: 0.8438
Epoch 19/30
618/618 [==============================] - 84s 136ms/step - loss: 0.0808 -
accuracy: 0.9714 - val_loss: 0.4435 - val_accuracy: 0.8593
Epoch 20/30
618/618 [==============================] - 84s 136ms/step - loss: 0.0698 -
accuracy: 0.9743 - val_loss: 0.4748 - val_accuracy: 0.8430
Epoch 21/30
618/618 [==============================] - 84s 137ms/step - loss: 0.0645 -
accuracy: 0.9763 - val_loss: 0.5240 - val_accuracy: 0.8542
Epoch 22/30
618/618 [==============================] - 84s 136ms/step - loss: 0.0602 -
accuracy: 0.9788 - val_loss: 0.5185 - val_accuracy: 0.8610
Epoch 23/30
618/618 [==============================] - 84s 136ms/step - loss: 0.0530 -
accuracy: 0.9818 - val_loss: 0.5937 - val_accuracy: 0.8533
Epoch 24/30
618/618 [==============================] - 84s 136ms/step - loss: 0.0502 -
accuracy: 0.9832 - val_loss: 0.5408 - val_accuracy: 0.8479
Epoch 25/30
618/618 [==============================] - 84s 136ms/step - loss: 0.0524 -
accuracy: 0.9819 - val_loss: 0.4855 - val_accuracy: 0.8527
Epoch 26/30
618/618 [==============================] - 84s 137ms/step - loss: 0.0493 -
accuracy: 0.9834 - val_loss: 0.5793 - val_accuracy: 0.8476
```

```
Epoch 27/30
618/618 [==============================] - 84s 137ms/step - loss: 0.0481 -
accuracy: 0.9829 - val_loss: 0.5886 - val_accuracy: 0.8473
Epoch 28/30
618/618 [==============================] - 84s 137ms/step - loss: 0.0462 -
accuracy: 0.9837 - val_loss: 0.6158 - val_accuracy: 0.8378
Epoch 29/30
618/618 [==============================] - 84s 136ms/step - loss: 0.0473 -
accuracy: 0.9843 - val_loss: 0.5764 - val_accuracy: 0.8430
Epoch 30/30
618/618 [==============================] - 84s 136ms/step - loss: 0.0438 -
accuracy: 0.9851 - val_loss: 0.7596 - val_accuracy: 0.8327
```

```
[24]: <keras.callbacks.History at 0x2975b446f10>
```

```
[25]: #ver las imagenes de la variable X sin modificaciones por aumento de datos
      plt.figure(figsize=(20, 8))
      for i in range(10):
        plt.subplot(2, 5, i+1)
        plt.xticks([])
        plt.yticks([])
        plt.imshow(X[i].reshape(100, 100), cmap="gray")
```

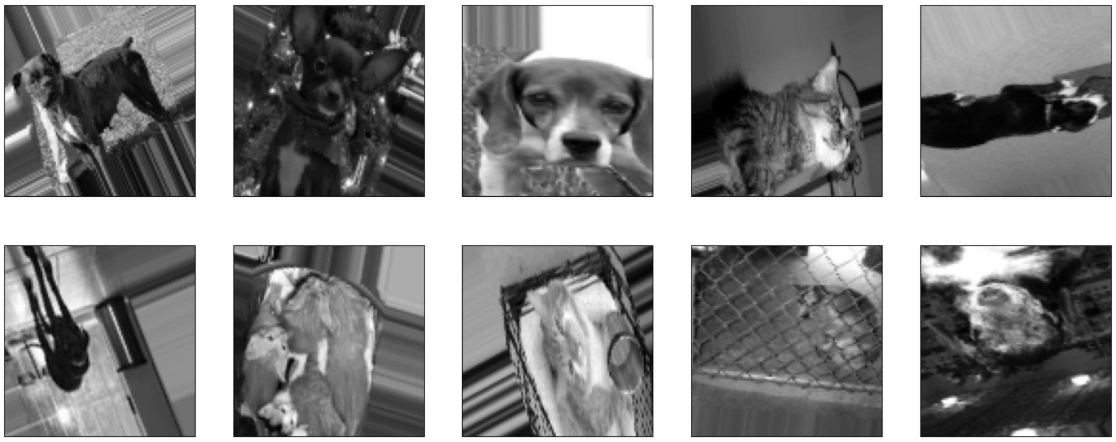
```
[26]: #Realizar el aumento de datos con varias transformaciones. Al final, graficar␣
       ↪10 como ejemplo
      from tensorflow.keras.preprocessing.image import ImageDataGenerator

      datagen = ImageDataGenerator(
          rotation_range=30,
          width_shift_range=0.2,
          height_shift_range=0.2,
          shear_range=15,
          zoom_range=[0.7, 1.4],
          horizontal_flip=True,
          vertical_flip=True
      )

      datagen.fit(X)

      plt.figure(figsize=(20,8))

      for imagen, etiqueta in datagen.flow(X, y, batch_size=10, shuffle=False):
        for i in range(10):
          plt.subplot(2, 5, i+1)
          plt.xticks([])
          plt.yticks([])
          plt.imshow(imagen[i].reshape(100, 100), cmap="gray")
        break
```



[27]:
```
modeloDenso_AD = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(100, 100, 1)),
  tf.keras.layers.Dense(150, activation='relu'),
  tf.keras.layers.Dense(150, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])

modeloCNN_AD = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(100, 100,␣
 ↪1)),
  tf.keras.layers.MaxPooling2D(2, 2),
  tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2, 2),
  tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2, 2),

  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(100, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])

modeloCNN2_AD = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(100, 100,␣
 ↪1)),
  tf.keras.layers.MaxPooling2D(2, 2),
  tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2, 2),
  tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2, 2),

  tf.keras.layers.Dropout(0.5),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(250, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])
```

[28]:
```
modeloDenso_AD.compile(optimizer='adam',
                    loss='binary_crossentropy',
                    metrics=['accuracy'])

modeloCNN_AD.compile(optimizer='adam',
                    loss='binary_crossentropy',
                    metrics=['accuracy'])

modeloCNN2_AD.compile(optimizer='adam',
                    loss='binary_crossentropy',
                    metrics=['accuracy'])
```

[29]:
```
#Separar los datos de entrenamiento y los datos de pruebas en variables␣
 ↪diferentes

len(X) * .85 #19700
len(X) - 19700 #3562

X_entrenamiento = X[:19700]
X_validacion = X[19700:]

y_entrenamiento = y[:19700]
y_validacion = y[19700:]
```

[30]:
```
#Usar la funcion flow del generador para crear un iterador que podamos enviar␣
↪como entrenamiento a la funcion FIT del modelo
data_gen_entrenamiento = datagen.flow(X_entrenamiento, y_entrenamiento,␣
↪batch_size=32)
```

[31]:
```
tensorboardDenso_AD = TensorBoard(log_dir='logs/denso_AD')

modeloDenso_AD.fit(
    data_gen_entrenamiento,
    epochs=25, batch_size=32,
    validation_data=(X_validacion, y_validacion),
    steps_per_epoch=int(np.ceil(len(X_entrenamiento) / float(32))),
    validation_steps=int(np.ceil(len(X_validacion) / float(32))),
    callbacks=[tensorboardDenso_AD]
)
```

```
Epoch 1/25
616/616 [==============================] - 18s 29ms/step - loss: 0.7347 -
accuracy: 0.5080 - val_loss: 0.6912 - val_accuracy: 0.5022
Epoch 2/25
616/616 [==============================] - 18s 29ms/step - loss: 0.6933 -
accuracy: 0.5025 - val_loss: 0.6924 - val_accuracy: 0.5157
Epoch 3/25
616/616 [==============================] - 18s 29ms/step - loss: 0.6935 -
accuracy: 0.5053 - val_loss: 0.6926 - val_accuracy: 0.5073
Epoch 4/25
616/616 [==============================] - 18s 29ms/step - loss: 0.6926 -
accuracy: 0.5027 - val_loss: 0.6931 - val_accuracy: 0.4986
Epoch 5/25
616/616 [==============================] - 18s 29ms/step - loss: 0.6928 -
accuracy: 0.4985 - val_loss: 0.6932 - val_accuracy: 0.4989
Epoch 6/25
616/616 [==============================] - 18s 29ms/step - loss: 0.6925 -
accuracy: 0.4992 - val_loss: 0.6931 - val_accuracy: 0.4997
Epoch 7/25
616/616 [==============================] - 18s 29ms/step - loss: 0.6922 -
accuracy: 0.5056 - val_loss: 0.6925 - val_accuracy: 0.5020
Epoch 8/25
616/616 [==============================] - 18s 28ms/step - loss: 0.6924 -
accuracy: 0.5090 - val_loss: 0.6931 - val_accuracy: 0.4994
Epoch 9/25
616/616 [==============================] - 18s 28ms/step - loss: 0.6930 -
accuracy: 0.5019 - val_loss: 0.6920 - val_accuracy: 0.5059
Epoch 10/25
616/616 [==============================] - 18s 29ms/step - loss: 0.6927 -
accuracy: 0.5042 - val_loss: 0.6916 - val_accuracy: 0.5059
Epoch 11/25
```

```
616/616 [==============================] - 18s 29ms/step - loss: 0.6912 -
accuracy: 0.5087 - val_loss: 0.6899 - val_accuracy: 0.5171
Epoch 12/25
616/616 [==============================] - 18s 29ms/step - loss: 0.6913 -
accuracy: 0.5126 - val_loss: 0.6921 - val_accuracy: 0.5020
Epoch 13/25
616/616 [==============================] - 18s 29ms/step - loss: 0.6921 -
accuracy: 0.5084 - val_loss: 0.6918 - val_accuracy: 0.5084
Epoch 14/25
616/616 [==============================] - 18s 28ms/step - loss: 0.6924 -
accuracy: 0.5046 - val_loss: 0.6914 - val_accuracy: 0.5020
Epoch 15/25
616/616 [==============================] - 18s 28ms/step - loss: 0.6919 -
accuracy: 0.5024 - val_loss: 0.6929 - val_accuracy: 0.4994
Epoch 16/25
616/616 [==============================] - 18s 29ms/step - loss: 0.6925 -
accuracy: 0.5005 - val_loss: 0.6930 - val_accuracy: 0.4989
Epoch 17/25
616/616 [==============================] - 18s 29ms/step - loss: 0.6920 -
accuracy: 0.5056 - val_loss: 0.6909 - val_accuracy: 0.5084
Epoch 18/25
616/616 [==============================] - 18s 29ms/step - loss: 0.6914 -
accuracy: 0.5095 - val_loss: 0.6903 - val_accuracy: 0.5020
Epoch 19/25
616/616 [==============================] - 18s 29ms/step - loss: 0.6919 -
accuracy: 0.5139 - val_loss: 0.6903 - val_accuracy: 0.5104
Epoch 20/25
616/616 [==============================] - 18s 29ms/step - loss: 0.6908 -
accuracy: 0.5105 - val_loss: 0.6895 - val_accuracy: 0.5261
Epoch 21/25
616/616 [==============================] - 18s 29ms/step - loss: 0.6908 -
accuracy: 0.5162 - val_loss: 0.6903 - val_accuracy: 0.5208
Epoch 22/25
616/616 [==============================] - 18s 29ms/step - loss: 0.6902 -
accuracy: 0.5096 - val_loss: 0.6907 - val_accuracy: 0.5098
Epoch 23/25
616/616 [==============================] - 18s 29ms/step - loss: 0.6911 -
accuracy: 0.5125 - val_loss: 0.6901 - val_accuracy: 0.5191
Epoch 24/25
616/616 [==============================] - 18s 29ms/step - loss: 0.6903 -
accuracy: 0.5187 - val_loss: 0.6904 - val_accuracy: 0.5211
Epoch 25/25
616/616 [==============================] - 18s 29ms/step - loss: 0.6897 -
accuracy: 0.5134 - val_loss: 0.6913 - val_accuracy: 0.5143
```

```
[31]: <keras.callbacks.History at 0x29760c4e4f0>
```

[32]:
```
tensorboardCNN_AD = TensorBoard(log_dir='logs-new/cnn_AD')

modeloCNN_AD.fit(
    data_gen_entrenamiento,
    epochs=40, batch_size=32,
    validation_data=(X_validacion, y_validacion),
    steps_per_epoch=int(np.ceil(len(X_entrenamiento) / float(32))),
    validation_steps=int(np.ceil(len(X_validacion) / float(32))),
    callbacks=[tensorboardCNN_AD]
)
```

```
Epoch 1/40
616/616 [==============================] - 79s 127ms/step - loss: 0.6814 -
accuracy: 0.5603 - val_loss: 0.6784 - val_accuracy: 0.5758
Epoch 2/40
616/616 [==============================] - 78s 126ms/step - loss: 0.6583 -
accuracy: 0.6110 - val_loss: 0.6997 - val_accuracy: 0.5924
Epoch 3/40
616/616 [==============================] - 78s 126ms/step - loss: 0.6465 -
accuracy: 0.6260 - val_loss: 0.6184 - val_accuracy: 0.6623
Epoch 4/40
616/616 [==============================] - 78s 127ms/step - loss: 0.6315 -
accuracy: 0.6419 - val_loss: 0.6370 - val_accuracy: 0.6364
Epoch 5/40
616/616 [==============================] - 78s 127ms/step - loss: 0.6244 -
accuracy: 0.6534 - val_loss: 0.5794 - val_accuracy: 0.6909
Epoch 6/40
616/616 [==============================] - 76s 124ms/step - loss: 0.6073 -
accuracy: 0.6655 - val_loss: 0.5889 - val_accuracy: 0.6746
Epoch 7/40
616/616 [==============================] - 73s 118ms/step - loss: 0.5889 -
accuracy: 0.6863 - val_loss: 0.5307 - val_accuracy: 0.7353
Epoch 8/40
616/616 [==============================] - 74s 120ms/step - loss: 0.5792 -
accuracy: 0.6993 - val_loss: 0.6081 - val_accuracy: 0.6496
Epoch 9/40
616/616 [==============================] - 73s 118ms/step - loss: 0.5690 -
accuracy: 0.7035 - val_loss: 0.5169 - val_accuracy: 0.7428
Epoch 10/40
616/616 [==============================] - 72s 117ms/step - loss: 0.5589 -
accuracy: 0.7135 - val_loss: 0.5013 - val_accuracy: 0.7580
Epoch 11/40
616/616 [==============================] - 72s 118ms/step - loss: 0.5474 -
accuracy: 0.7231 - val_loss: 0.5357 - val_accuracy: 0.7291
Epoch 12/40
616/616 [==============================] - 72s 118ms/step - loss: 0.5398 -
accuracy: 0.7263 - val_loss: 0.4919 - val_accuracy: 0.7597
```

```
Epoch 13/40
616/616 [==============================] - 73s 118ms/step - loss: 0.5294 -
accuracy: 0.7336 - val_loss: 0.4610 - val_accuracy: 0.7760
Epoch 14/40
616/616 [==============================] - 72s 117ms/step - loss: 0.5190 -
accuracy: 0.7423 - val_loss: 0.4756 - val_accuracy: 0.7726
Epoch 15/40
616/616 [==============================] - 73s 119ms/step - loss: 0.5117 -
accuracy: 0.7472 - val_loss: 0.4328 - val_accuracy: 0.8004
Epoch 16/40
616/616 [==============================] - 74s 120ms/step - loss: 0.5094 -
accuracy: 0.7509 - val_loss: 0.4435 - val_accuracy: 0.7948
Epoch 17/40
616/616 [==============================] - 72s 117ms/step - loss: 0.5008 -
accuracy: 0.7560 - val_loss: 0.4433 - val_accuracy: 0.7911
Epoch 18/40
616/616 [==============================] - 73s 118ms/step - loss: 0.4917 -
accuracy: 0.7622 - val_loss: 0.4862 - val_accuracy: 0.7726
Epoch 19/40
616/616 [==============================] - 73s 118ms/step - loss: 0.4792 -
accuracy: 0.7688 - val_loss: 0.3876 - val_accuracy: 0.8265
Epoch 20/40
616/616 [==============================] - 73s 118ms/step - loss: 0.4832 -
accuracy: 0.7684 - val_loss: 0.3965 - val_accuracy: 0.8186
Epoch 21/40
616/616 [==============================] - 73s 118ms/step - loss: 0.4774 -
accuracy: 0.7707 - val_loss: 0.4189 - val_accuracy: 0.8091
Epoch 22/40
616/616 [==============================] - 73s 118ms/step - loss: 0.4694 -
accuracy: 0.7774 - val_loss: 0.4935 - val_accuracy: 0.7698
Epoch 23/40
616/616 [==============================] - 73s 118ms/step - loss: 0.4661 -
accuracy: 0.7800 - val_loss: 0.4575 - val_accuracy: 0.7841
Epoch 24/40
616/616 [==============================] - 74s 120ms/step - loss: 0.4638 -
accuracy: 0.7810 - val_loss: 0.3866 - val_accuracy: 0.8245
Epoch 25/40
616/616 [==============================] - 73s 118ms/step - loss: 0.4572 -
accuracy: 0.7824 - val_loss: 0.4054 - val_accuracy: 0.8212
Epoch 26/40
616/616 [==============================] - 72s 118ms/step - loss: 0.4608 -
accuracy: 0.7834 - val_loss: 0.3802 - val_accuracy: 0.8262
Epoch 27/40
616/616 [==============================] - 73s 118ms/step - loss: 0.4533 -
accuracy: 0.7851 - val_loss: 0.4001 - val_accuracy: 0.8206
Epoch 28/40
616/616 [==============================] - 73s 118ms/step - loss: 0.4456 -
accuracy: 0.7895 - val_loss: 0.3859 - val_accuracy: 0.8304
```

```
Epoch 29/40
616/616 [==============================] - 73s 118ms/step - loss: 0.4420 -
accuracy: 0.7948 - val_loss: 0.4361 - val_accuracy: 0.7987
Epoch 30/40
616/616 [==============================] - 73s 118ms/step - loss: 0.4430 -
accuracy: 0.7935 - val_loss: 0.3731 - val_accuracy: 0.8321
Epoch 31/40
616/616 [==============================] - 73s 119ms/step - loss: 0.4441 -
accuracy: 0.7874 - val_loss: 0.4080 - val_accuracy: 0.8105
Epoch 32/40
616/616 [==============================] - 73s 118ms/step - loss: 0.4331 -
accuracy: 0.7981 - val_loss: 0.4907 - val_accuracy: 0.7777
Epoch 33/40
616/616 [==============================] - 73s 118ms/step - loss: 0.4355 -
accuracy: 0.7958 - val_loss: 0.4181 - val_accuracy: 0.8088
Epoch 34/40
616/616 [==============================] - 73s 118ms/step - loss: 0.4304 -
accuracy: 0.7995 - val_loss: 0.3641 - val_accuracy: 0.8352
Epoch 35/40
616/616 [==============================] - 73s 118ms/step - loss: 0.4274 -
accuracy: 0.8012 - val_loss: 0.3998 - val_accuracy: 0.8150
Epoch 36/40
616/616 [==============================] - 72s 117ms/step - loss: 0.4175 -
accuracy: 0.8089 - val_loss: 0.3819 - val_accuracy: 0.8243
Epoch 37/40
616/616 [==============================] - 72s 117ms/step - loss: 0.4227 -
accuracy: 0.8035 - val_loss: 0.5207 - val_accuracy: 0.7661
Epoch 38/40
616/616 [==============================] - 72s 118ms/step - loss: 0.4189 -
accuracy: 0.8084 - val_loss: 0.3683 - val_accuracy: 0.8316
Epoch 39/40
616/616 [==============================] - 73s 119ms/step - loss: 0.4166 -
accuracy: 0.8070 - val_loss: 0.3445 - val_accuracy: 0.8467
Epoch 40/40
616/616 [==============================] - 73s 119ms/step - loss: 0.4104 -
accuracy: 0.8085 - val_loss: 0.3727 - val_accuracy: 0.8394
```

[32]:
```
<keras.callbacks.History at 0x29760d26be0>
```

[33]:
```python
tensorboardCNN2_AD = TensorBoard(log_dir='logs/cnn2_AD')

modeloCNN2_AD.fit(
    data_gen_entrenamiento,
    epochs=40, batch_size=32,
    validation_data=(X_validacion, y_validacion),
    steps_per_epoch=int(np.ceil(len(X_entrenamiento) / float(32))),
    validation_steps=int(np.ceil(len(X_validacion) / float(32))),
    callbacks=[tensorboardCNN2_AD]
)
```

```
Epoch 1/40
616/616 [==============================] - 82s 132ms/step - loss: 0.6854 -
accuracy: 0.5512 - val_loss: 0.6727 - val_accuracy: 0.6157
Epoch 2/40
616/616 [==============================] - 81s 131ms/step - loss: 0.6711 -
accuracy: 0.5904 - val_loss: 0.6419 - val_accuracy: 0.6255
Epoch 3/40
616/616 [==============================] - 81s 132ms/step - loss: 0.6574 -
accuracy: 0.6110 - val_loss: 0.6531 - val_accuracy: 0.6154
Epoch 4/40
616/616 [==============================] - 81s 132ms/step - loss: 0.6398 -
accuracy: 0.6312 - val_loss: 0.6183 - val_accuracy: 0.6637
Epoch 5/40
616/616 [==============================] - 81s 132ms/step - loss: 0.6301 -
accuracy: 0.6465 - val_loss: 0.5815 - val_accuracy: 0.7024
Epoch 6/40
616/616 [==============================] - 81s 132ms/step - loss: 0.6193 -
accuracy: 0.6586 - val_loss: 0.6123 - val_accuracy: 0.6463
Epoch 7/40
616/616 [==============================] - 81s 132ms/step - loss: 0.6052 -
accuracy: 0.6732 - val_loss: 0.5612 - val_accuracy: 0.7019
Epoch 8/40
616/616 [==============================] - 81s 131ms/step - loss: 0.6031 -
accuracy: 0.6717 - val_loss: 0.5563 - val_accuracy: 0.7193
Epoch 9/40
616/616 [==============================] - 81s 131ms/step - loss: 0.5916 -
accuracy: 0.6840 - val_loss: 0.5854 - val_accuracy: 0.6856
Epoch 10/40
616/616 [==============================] - 81s 132ms/step - loss: 0.5857 -
accuracy: 0.6885 - val_loss: 0.5471 - val_accuracy: 0.7285
Epoch 11/40
616/616 [==============================] - 81s 132ms/step - loss: 0.5776 -
accuracy: 0.6960 - val_loss: 0.5504 - val_accuracy: 0.7142
Epoch 12/40
616/616 [==============================] - 81s 132ms/step - loss: 0.5726 -
accuracy: 0.7028 - val_loss: 0.5475 - val_accuracy: 0.7190
Epoch 13/40
616/616 [==============================] - 81s 131ms/step - loss: 0.5683 -
accuracy: 0.7050 - val_loss: 0.5238 - val_accuracy: 0.7403
Epoch 14/40
616/616 [==============================] - 81s 132ms/step - loss: 0.5660 -
accuracy: 0.7098 - val_loss: 0.5136 - val_accuracy: 0.7546
Epoch 15/40
616/616 [==============================] - 81s 132ms/step - loss: 0.5558 -
```

```
accuracy: 0.7173 - val_loss: 0.5192 - val_accuracy: 0.7468
Epoch 16/40
616/616 [==============================] - 81s 132ms/step - loss: 0.5572 -
accuracy: 0.7142 - val_loss: 0.5278 - val_accuracy: 0.7375
Epoch 17/40
616/616 [==============================] - 81s 132ms/step - loss: 0.5507 -
accuracy: 0.7224 - val_loss: 0.4954 - val_accuracy: 0.7633
Epoch 18/40
616/616 [==============================] - 81s 132ms/step - loss: 0.5451 -
accuracy: 0.7243 - val_loss: 0.4809 - val_accuracy: 0.7678
Epoch 19/40
616/616 [==============================] - 81s 132ms/step - loss: 0.5400 -
accuracy: 0.7261 - val_loss: 0.5536 - val_accuracy: 0.7322
Epoch 20/40
616/616 [==============================] - 81s 132ms/step - loss: 0.5398 -
accuracy: 0.7293 - val_loss: 0.4960 - val_accuracy: 0.7631
Epoch 21/40
616/616 [==============================] - 81s 132ms/step - loss: 0.5342 -
accuracy: 0.7322 - val_loss: 0.4651 - val_accuracy: 0.7793
Epoch 22/40
616/616 [==============================] - 82s 132ms/step - loss: 0.5303 -
accuracy: 0.7363 - val_loss: 0.4601 - val_accuracy: 0.7906
Epoch 23/40
616/616 [==============================] - 81s 132ms/step - loss: 0.5261 -
accuracy: 0.7366 - val_loss: 0.4586 - val_accuracy: 0.7931
Epoch 24/40
616/616 [==============================] - 81s 132ms/step - loss: 0.5267 -
accuracy: 0.7377 - val_loss: 0.4921 - val_accuracy: 0.7617
Epoch 25/40
616/616 [==============================] - 82s 133ms/step - loss: 0.5160 -
accuracy: 0.7488 - val_loss: 0.4832 - val_accuracy: 0.7706
Epoch 26/40
616/616 [==============================] - 81s 132ms/step - loss: 0.5170 -
accuracy: 0.7447 - val_loss: 0.5252 - val_accuracy: 0.7406
Epoch 27/40
616/616 [==============================] - 82s 133ms/step - loss: 0.5113 -
accuracy: 0.7472 - val_loss: 0.4646 - val_accuracy: 0.7821
Epoch 28/40
616/616 [==============================] - 82s 133ms/step - loss: 0.5101 -
accuracy: 0.7525 - val_loss: 0.5106 - val_accuracy: 0.7358
Epoch 29/40
616/616 [==============================] - 82s 132ms/step - loss: 0.5084 -
accuracy: 0.7489 - val_loss: 0.4650 - val_accuracy: 0.7925
Epoch 30/40
616/616 [==============================] - 82s 133ms/step - loss: 0.5026 -
accuracy: 0.7573 - val_loss: 0.4434 - val_accuracy: 0.8010
Epoch 31/40
616/616 [==============================] - 81s 132ms/step - loss: 0.5095 -
```

```
accuracy: 0.7480 - val_loss: 0.4373 - val_accuracy: 0.8063
Epoch 32/40
616/616 [==============================] - 82s 133ms/step - loss: 0.4956 -
accuracy: 0.7580 - val_loss: 0.5163 - val_accuracy: 0.7521
Epoch 33/40
616/616 [==============================] - 82s 133ms/step - loss: 0.4987 -
accuracy: 0.7558 - val_loss: 0.4378 - val_accuracy: 0.7987
Epoch 34/40
616/616 [==============================] - 83s 135ms/step - loss: 0.4935 -
accuracy: 0.7617 - val_loss: 0.5073 - val_accuracy: 0.7527
Epoch 35/40
616/616 [==============================] - 82s 132ms/step - loss: 0.4953 -
accuracy: 0.7621 - val_loss: 0.4243 - val_accuracy: 0.8049
Epoch 36/40
616/616 [==============================] - 82s 133ms/step - loss: 0.4912 -
accuracy: 0.7630 - val_loss: 0.4521 - val_accuracy: 0.7920
Epoch 37/40
616/616 [==============================] - 82s 134ms/step - loss: 0.4880 -
accuracy: 0.7649 - val_loss: 0.4726 - val_accuracy: 0.7774
Epoch 38/40
616/616 [==============================] - 81s 132ms/step - loss: 0.4845 -
accuracy: 0.7669 - val_loss: 0.4345 - val_accuracy: 0.8035
Epoch 39/40
616/616 [==============================] - 82s 133ms/step - loss: 0.4834 -
accuracy: 0.7671 - val_loss: 0.4467 - val_accuracy: 0.7866
Epoch 40/40
616/616 [==============================] - 83s 135ms/step - loss: 0.4819 -
accuracy: 0.7684 - val_loss: 0.4369 - val_accuracy: 0.7945
```

[33]:
```
<keras.callbacks.History at 0x29761310ee0>
```

[35]:
```
modeloDenso.save('perros-gatos-denso_V3.h5')
modeloDenso_AD.save('perros-gatos-denso-ad_V3.h5')

modeloCNN_AD.save('perros-gatos-cnn-ad_V3.h5')
modeloCNN.save('perros-gatos-cnn_V3.h5')

modeloCNN2_AD.save('perros-gatos-cnn2-ad_V3.h5')
modeloCNN2.save('perros-gatos-cnn2_V3.h5')
```

[186]:
```
%load_ext tensorboard
%tensorboard --logdir logs
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

```
Reusing TensorBoard on port 6006 (pid 9060), started 1:30:21 ago. (Use '!kill␣
↪9060' to kill it.)
```

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

[129]:
```
import tensorflow as tf
import tensorflow_datasets as tfds
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import confusion_matrix, f1_score, roc_curve,␣
 ↪precision_score, recall_score, accuracy_score, roc_auc_score
from sklearn import metrics
#from mlxtend.plotting import plot_confusion_matrix
from keras.models import load_model
from keras.preprocessing.image import ImageDataGenerator
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
ModelF= load_model('perros-gatos-cnn_V1.h5')
#ModelF= load_model('perros-gatos-denso-ad_V3.h5')
```

[194]:
```
import numpy as np
#from google.colab import files
from keras.preprocessing import image
import keras
import tensorflow as tf

#uploaded = files.upload()
import matplotlib.pyplot as plt
import cv2

plt.figure(figsize=(10,10))
imagen = cv2.imread('ImgPre/per8.jpg')
plt.subplot(1, 2, 1)
plt.xticks([])
plt.yticks([])
plt.imshow(imagen, cmap='gray')
plt.title("Original")


TAMANO_IMG=100

imagen = cv2.resize(imagen, (TAMANO_IMG, TAMANO_IMG))
imagen = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)
plt.subplot(1, 2, 2)
plt.xticks([])
plt.yticks([])
plt.imshow(imagen, cmap='gray')
plt.title("Para el modelo")
```

```
  # predicting images
img=imagen
  #img = keras.utils.load_img(path, target_size=(100, 100))
x = keras.utils.img_to_array(img)
x = np.expand_dims(x, axis=0)

images = np.vstack([x])
classes = ModelF.predict(images, batch_size=10)
print("retorno de la predicción = ",classes[0])
if classes[0]>0.5:
    print(" ** ES PERRO  **")
else:
    print(" ** ES GATO  **")
```

```
1/1 [==============================] - 0s 20ms/step
retorno de la predicción =  [1.]
 ** ES PERRO  **
```

```
[ ]:
```