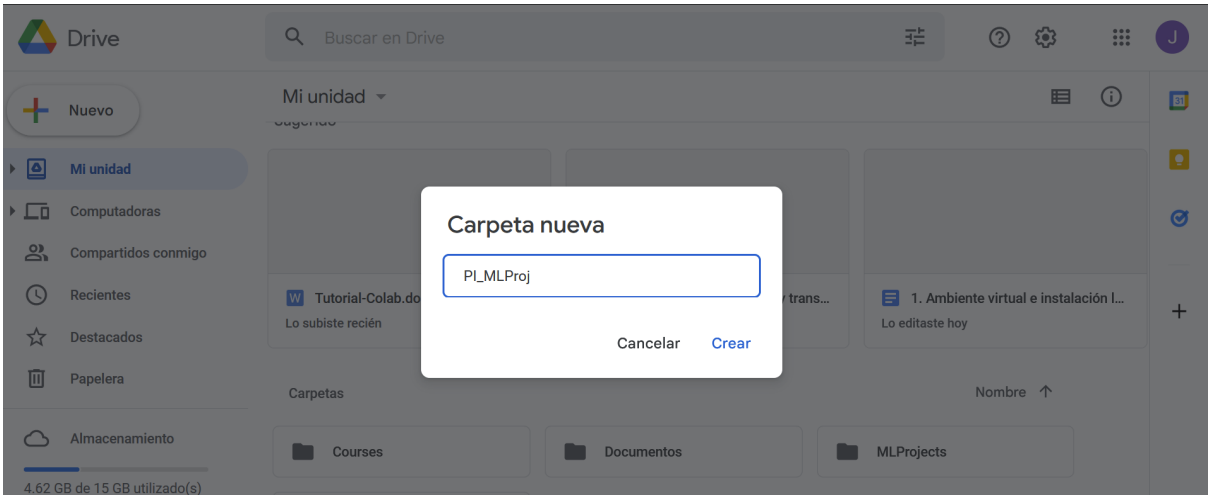
En este segundo paso entrenará el modelo de aprendizaje de máquina utilizando como base un modelo pre-entrenado (Transferencia de aprendizaje). Para esto, utilizará Google colab.

1. Instalación de Google colab

Si nunca ha utilizado Google colaboratory, debe instalarlo. Para esto debe seguir los pasos del tutorial Tutorial-colab

2. Crear carpeta para el proyecto

Tendrá que crear una carpeta para en Google drive para el proyecto. Para esto, diríjase a Nuevo -> Carpeta nueva y escriba el nombre de la carpeta, en este casi PI_MLProject (no tiene que ser igual al nombre de la carpeta del ambiente virtual)

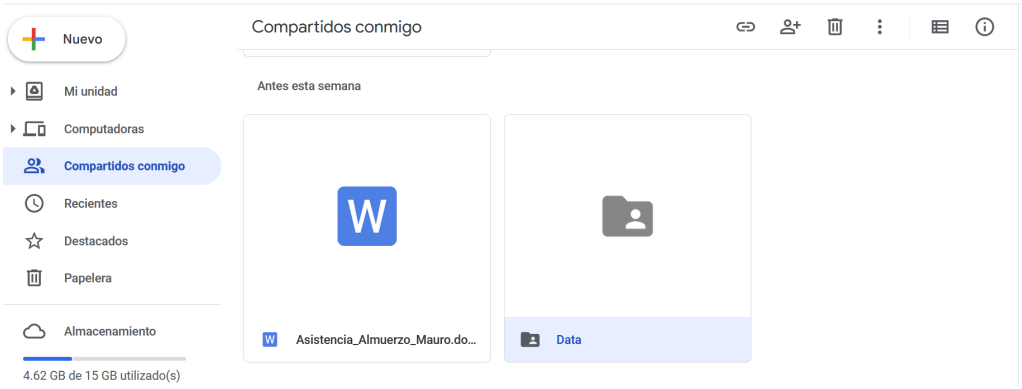


Finalmente, entre a la carpeta.

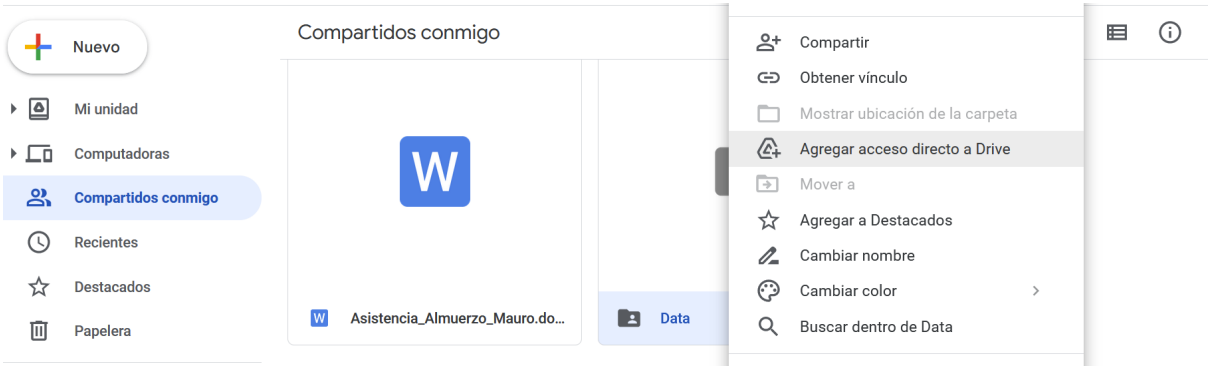
3. Acceso a los datos

https://drive.google.com/drive/folders/1w7cEuSVij8V87-wrHHlLm-QorRgCL8BI?usp=sharing

Cuando acceda a este vínculo, se creará en “Compartidos conmigo” de Google Drive un acceso directo a una carpeta **Data**. **Nota:** Tengan en cuenta que en ningún momento tendremos que descargar la dicha carpeta.



Cree un acceso directo a la carpeta Data desde la carpeta del proyecto. Para esto haga clic derecho en la carpeta Data, y después haga clic en la opción Agregar acceso directo a Drive.



Se desplegrá un menú de búsqueda en el que tendrá que buscar la carpeta del proyecto PI_MLProject. Si creó la carpeta en la raíz de Drive, la encontrará en:

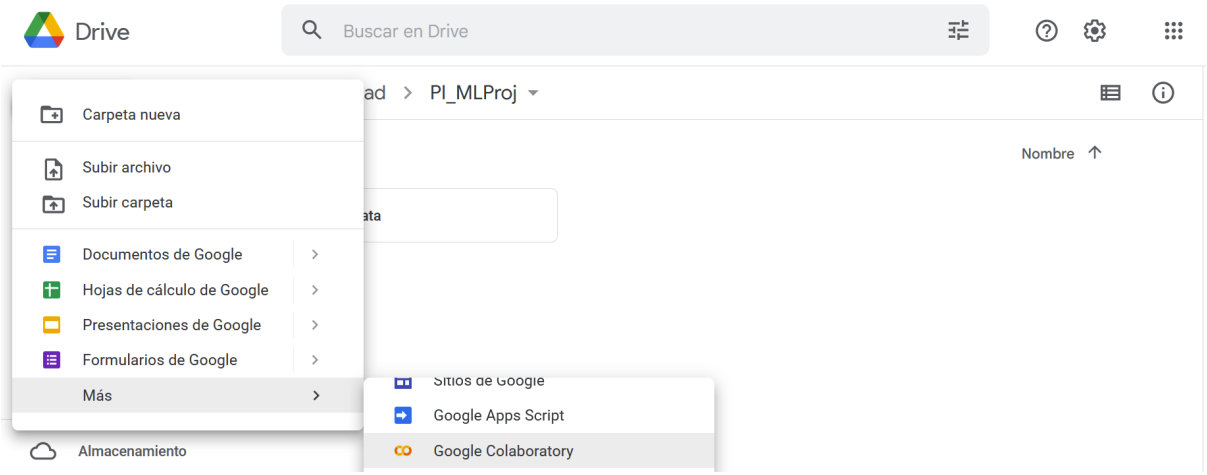
Mi unidad/PI_MLProject

En la carpeta raíz del proyecto encontrará
los siguientes directorios:

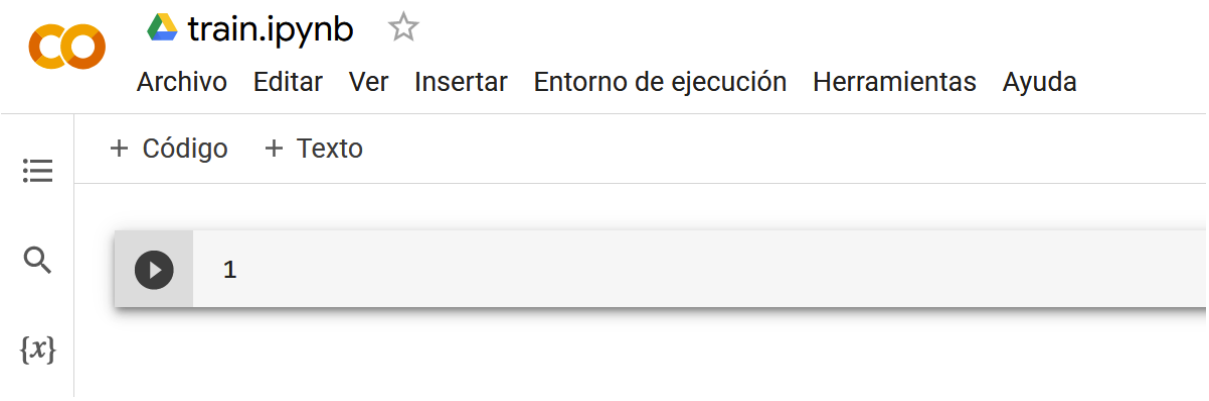
Data/cats_vs_dogs_smal/train
Data/cats_vs_dogs_small/test

4. Crear archivo de Google colaboratory para entrenar nuesto modelo

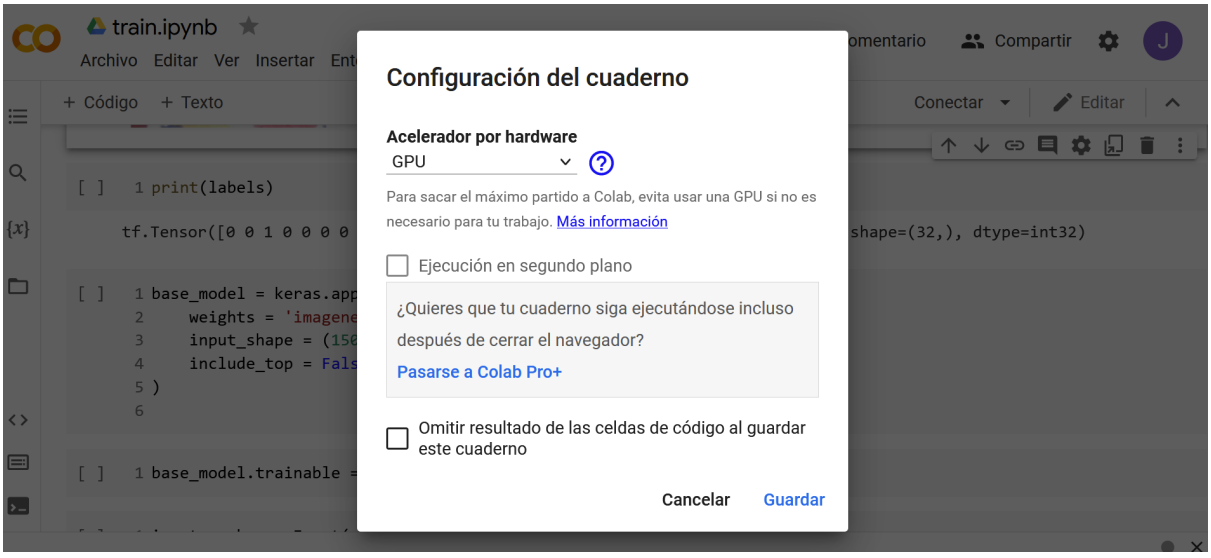
En la carpeta PI_MLProject diríjase a Nuevo -> Más -> Google colaboratory



Esto abrirá en una pestaña nueva el documento Untitled0.ipynb. Podrá cambiar el nombre de este archivo a train.ipynb (No es necesario, es por convención).



Dado que va a entrenar una red neuronal, debe activar la GPU que proporciona Google colaboratory. Para esto vaya a Entorno de ejecución -> Cambiar tipo de entorno de ejecución. En el menú que se despliega, en la parte Acelerador por hardware, cambie de None a GPU.



5. Vincular Google drive con Google colaboratory

Para tener acceso a los datos que tiene almacenados en Google drive desde el documento .ipynb de Google colaboratory, tendrá que ejecutar en una celda las siguientes líneas de código:

```
from google.colab import drive
drive.mount('/content/drive')
```

A pesar de que el documento está creado en la carpeta del proyecto, si ejecuta en una celda el comando

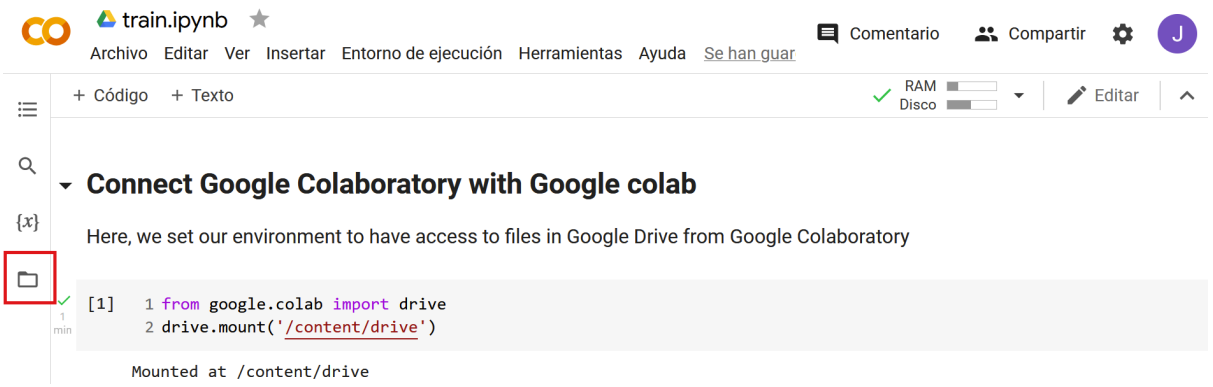
```
!pwd
```

Se dará cuenta de que el *working directory* no corresponde a la carpeta del proyecto. Para movernos hasta esta carpeta, tendrá que ejecutar en una celda lo siguiente:

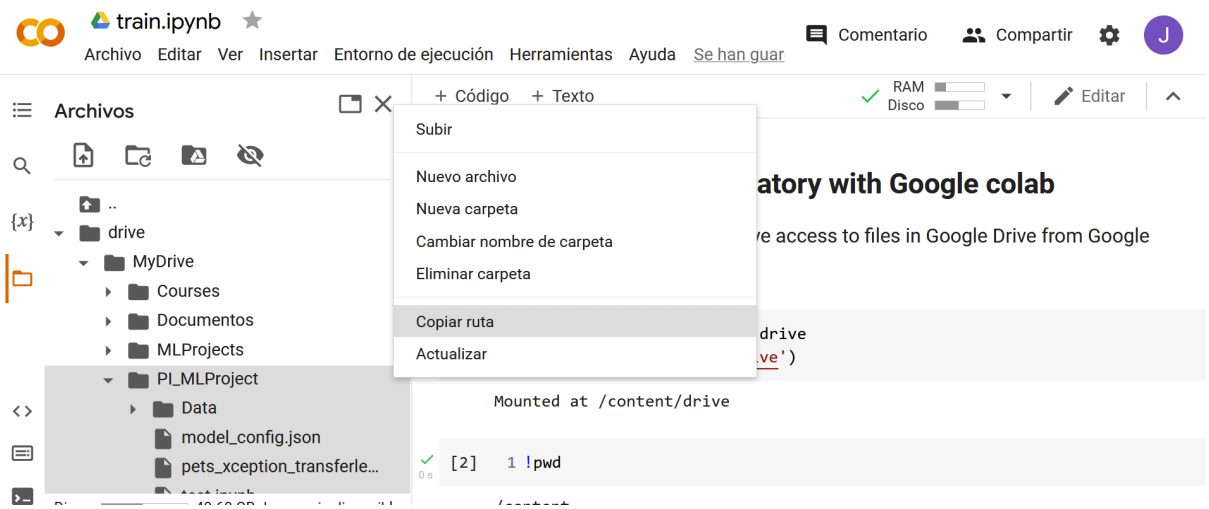
```
cd '/content/drive/MyDrive/PI_MLProject'
```

Nota: El path varía dependiendo de la ubicación en la que se haya creado la carpeta

Para tener una idea de cómo navegar en los directorios de Google drive, puede abrir la carpeta de navegación:



Haciendo clic derecho en los tres puntos que aparecen cuando ponemos el cursor encima de una carpeta, tendrá acceso al path completo de dicha carpeta.



En colab, cuando escribe un comando con !, este se entiende como un comando bash, de lo contrario se ejecuta como un comando Python.

6. Acceso a los datos desde Google colaboratory

Dado que ya cuenta con un acceso directo a la carpeta de datos desde Google drive, lo único que tiene que hacer es definir en una variable el path de dicha carpeta. Para esto, ejecute en una celda nueva la siguiente linea:

```
data_path =
'/content/drive/MyDrive/PI_MLProject/Data/cats_vs_dogs_small'
```

7. Entrenamiento del modelo

Finalmente, entrenará el modelo de clasificación de mascotas (perros vs gatos). Lo primero que debe hacer es definir las librerías que va a utilizar. En este caso se utilizará keras y tensorflow dado que es uno de los paquetes para implementación de redes neuronales más utilizado. Sin embargo, el mismo procesidimiento se puede seguir utilizando otros paquetes como Pytorch, MxNet, FastAI, JAX.

En una celda importer los siguientes módulos:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing import image_dataset_from_directory
```

Para saber en qué versiones de las librerías está trabajando, puede ejecutar las siguientes instrucciones:

```
print(tf.keras.__version__)
print(tf.__version__)
```

Esto puede ser útil a la hora de crear ambientes virtuales (local) para que no se presente ningún tipo de incompatibilidad.

Importe las siguientes librerías adicionales que le ayudarán para el manejo de imágenes, arreglos numéricos, gráficas y directorios:

```
from PIL import Image                # to load images
from IPython.display import display # to display images
import matplotlib.pyplot as plt
import numpy as np
import os
```

Grafique una de las imágenes del conjunto de datos de entrenamiento:

En una celda, defina el path de dicha imagen:

```
set_name = 'train'
class_name = "dog"
file_name = 'dog.1.jpg'
file_path = os.path.join(data_path,set_name,class_name,file_name)
print(file_path)
```

Después de tener el path, en otra celda mostramos la imagenr:

```
img = Image.open(file_path)
display(img)
img_array = np.array(img)
print(img_array.shape)
```

Adicional a la imágen, está imprimiendo su tamaño, en este caso (499, 327, 3). Esta dimensión se puede ver como (alto x ancho x canales (RGB)). Si grafica diferentes imágenes, se dará cuenta de que el tamaño de cada una de ellas es diferente. Esto es algo que tendrá que organizar antes de enviarlas a la red neuronal.

Después, en otra celda debe dividir los datos en entrenamiento y validación. Esto se hace para que el modelo aprenda a generalizar (funcione bien ante imágenes no utilizadas en el entrenamiento). Esto se hace utilizando ‘dataloaders’. Por lo general, todos los módulos de redes neuronales tienen forma de crear dataloaders a partir de cualquier tipo de dato. De forma resumida, los dataloaders almancenan los directorios de cada una una de las imágenes de la base de datos y los organizan en batches (grupo) de tamaño batch_size. Esto hace que en el entrenamiento, carguemos en memoria solo grupos de 32 imágenes

(en este caso) y no las más de 25000 imágenes que tenemos, para lo que necesitaríamos mucha más memoria de la disponible.

```
training_path = os.path.join(data_path,'train')
training_set = image_dataset_from_directory(training_path,
                                            shuffle=True,
                                            batch_size=32,
                                            image_size=(150, 150),
                                            validation_split = 0.2,
                                            subset = 'training',
                                            seed = 1234,
                                            )
validation_set = image_dataset_from_directory(training_path,
                                            shuffle=True,
                                            batch_size=32,
                                            image_size=(150, 150),
                                            validation_split = 0.2,
                                            subset = 'validation',
                                            seed = 1234,
                                            )
```

Note que uno de los parámetros es image_size = (150,150). Esto hace que cada imagen se procese al tamaño (150,150,3) antes de ser entregada a la red neuronal.

Para saber en qué clases se van a clasificar las imágenes de la base de datos, puede utilizar la siguiente instrucción:

```
training_set.class_names
```

Los nombres [‘cat’, ‘dog’] salen del nombre de las carpetas en el directorio de datos. Es decir, debe tener una carpeta con imágenes de cada una de las clases que quiere separar.

En otra celda va a graficar algunas de las imágenes en la base de datos:

```
class_names = training_set.class_names
plt.figure(figsize=(10, 10))
for images, labels in training_set.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
        plt.axis("off")
```

Después de organizar los datos, va a definir el modelo de red neuronal que va a utilizar. En una celda, escriba las siguientes líneas:

```
base_model = keras.applications.Xception(
    weights = 'imagenet',
    input_shape = (150,150,3),
    include_top = False,
)
base_model.trainable = False
```

Acá se está definiendo que se va a utilizar como modelo base la arquitectura de red neuronal Xception pre-entrenada en la base de datos 'imagenet'. Esta parte servirá como extracción de características de alto nivel. Después, se define que la parte de la red que clasifica (include_top) no la vamos a utilizar. Es decir, se creará un clasificador propio. También se define que los pesos de esta arquitectura base (Xception) no se van a modificar.

Ahora se definirá el grafo computacional del clasificadro. En una celda nueva escriba las siguientes instrucciones:

```
inputs = keras.Input(shape = (150,150,3))
x = tf.keras.applications.xception.preprocess_input(inputs)
x = base_model(x, training=False)
x = keras.layers.GlobalAveragePooling2D()(x)
x = keras.layers.Dropout(0.2)(x)
outputs = keras.layers.Dense(1)(x)
model = keras.Model(inputs,outputs)
```

Acá se define lo siguiente:

Las entradas al modelo serán imágenes de tamaño (150,150,3). Debe coincidir con el parámetro input_shape definido en la etapa anterior.

Las entradas (inputs) serán preprocesadas (los pixeles pasan de ser valores entre 0-255 a ser valores entre -1 y 1). Este rango numérico es más fácil de procesar para la red neuronal. Estos datos pre-procesados se almacenan en una variable genérica x.

Se define que la primera parte del modelo será el base_model (xception pre-entrenado en imagenet) y que sus parámetros no se deberán modificar. (Esta parte es redundante dado que ya se había definido anteriormente, sin embargo, se hace por asegurar que no haya entrenamiento en estos parámetros).

Las características que entrega el base_model se promedian, consevando el tamaño del batch y el número de canales. Esto se hace para generar un vector de características que se pueda entregar al clasificador.

Se agrega una capa dropout para regularizar el modelo (ayuda a generalizar - tópicos avanzados de aprendizaje de máquina).

Se define la salida del modelo, en este caso, la probabilidad de que la imagen que se entregue sea de la clase *"perro".*

Se define el modelo completo (grafo computacional).

Finalmente, se entrena el modelo:

```
model.compile(optimizer='adam', loss = tf.keras.losses.BinaryCrossentropy(from_logits = True),metrics = keras.metrics.BinaryAccuracy())
model.fit(training_set, epochs = 20, validation_data = validation_set)
```

Para desplegar el modelo, tendrá que almacenar sus parámetros después del entrenamiento. Para esto, ejecute las siguientes líneas:

```
json_config = model.to_json()
with open('model_config.json', 'w') as json_file:
    json_file.write(json_config)

model.save_weights('pets_xception_transferlearning.h5')
```

En el archivo .json quedará almacenada la arquitectura de la red neuronal mientras que en el archivo pets_xception_transferlearning.h5 quedarán almacenados todos los pesos del modelo. Estos son los archivos que necesitará para desplegar el modelo. Estos archivos quedan almacenados en la carpeta raíz del proyecto.