```
from google.colab import drive
drive.mount('/gdrive/')
%cd /gdrive
```

```
Mounted at /gdrive/
/gdrive
```

```
ls
```

```
MyDrive/  Shareddrives/
```

```
cd/gdrive/My Drive/Multinomial animal classification/
```

```
/gdrive/My Drive/Multinomial animal classification
```

```
ls
```

```
cats/  dogs/  horses/
```

```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np
from tensorflow.keras.layers.experimental import preprocessing
from tensorflow.keras.preprocessing.image import load_img
```

```
import os
import PIL
import pathlib
import pandas as pd
import numpy as np
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import preprocessing
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
```

```
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.python.ops.numpy_ops import np_utils

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


BATCH_SIZE = 62
IMAGE_SIZE = 256
EPOCHS=50
CHANNELS=3


dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "/gdrive/My Drive/Multinomial animal classification/",
    seed=123,
    shuffle=True,
    image_size=(IMAGE_SIZE,IMAGE_SIZE),
    batch_size=BATCH_SIZE
)
```

```
Found 606 files belonging to 3 classes.
```

```
class_names = dataset.class_names
class_names
```

```
['cats', 'dogs', 'horses']
```

```
len(dataset)
```

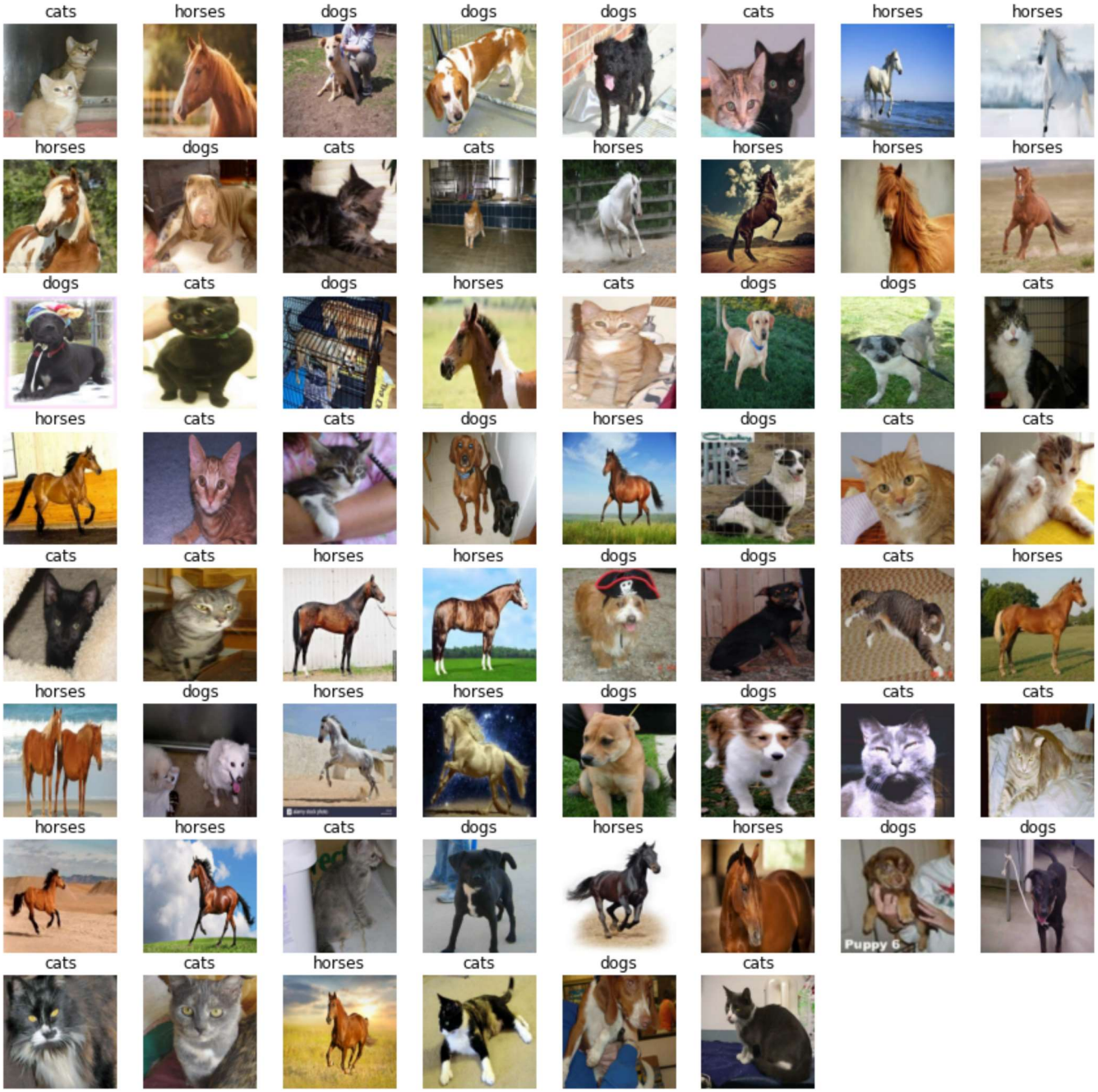
```
10
```

```
for image_batch, label_batch in dataset.take(1):
  print(image_batch.shape)
  print(image_batch[1])
  print(label_batch.numpy())
```

```
(62, 256, 256, 3)
tf.Tensor(
[[[ 17.091797  13.091797  28.091797]
  [ 33.476562  30.214844  44.845703]
  [ 39.691406  37.691406  51.691406]
  ...
  [188.69336  161.07812  140.23242 ]
  [190.47656  166.63086  146.36914 ]
  [186.36914  163.49219  146.87695 ]]

 [[ 51.65497   47.65497   62.65497 ]
  [ 56.120422  52.858704  67.48956 ]
  [ 45.33954   43.33954   57.33954 ]
```

```
         ...
         [189.45978  161.84454  140.99884 ]
         [188.9986   165.1529   144.89117 ]
         [185.35138  162.47443  144.26971 ]]

        [[ 73.170044  69.170044  84.170044]
         [ 65.42218   61.537537  76.168396]
         [ 36.514038  32.826538  46.826538]
         ...
         [195.31512  167.69989  146.85419 ]
         [193.06445  169.21875  148.95703 ]
         [190.20477  167.32782  148.95868 ]]

        ...

        [[135.9151   146.07135  148.07135 ]
         [145.86835  155.344    157.65546 ]
         [142.30444  150.42468  153.36456 ]
         ...
         [ 39.55902   37.71527   23.934021]
         [ 37.1474    36.900513  21.5224  ]
         [ 46.068054  46.755554  30.443054]]

        [[128.05878  139.05878  141.05878 ]
         [141.04938  151.68024  153.68024 ]
         [141.09961  149.9845   152.54205 ]
         ...
         [ 39.854553  38.739197  20.724731]
         [ 37.339294  36.339294  18.151794]
         [ 45.71466   44.71466   26.52716 ]]

        [[114.353516 125.353516 127.353516]
         [126.21484  136.8457   138.8457  ]
         [125.69336  135.69336  137.69336 ]
         ...
         [ 29.53711   27.30664    9.152344]
         [ 25.892578  24.892578   4.892578]
         [ 33.13867   32.13867   12.138672]]], shape=(256, 256, 3), dtype=float32)
       [1 1 1 0 1 1 1 1 0 2 1 0 0 2 0 1 1 1 0 0 2 0 1 2 2 1 0 2 0 2 2 1 2 2 2 2 1
        0 0 0 1 0 1 2 1 0 2 2 0 2 2 2 1 0 1 2 0 2 2 2 1 1]
```

```
plt.figure(figsize=(15, 15))
for image_batch, labels_batch in dataset.take(1):
    for i in range(BATCH_SIZE):
        ax = plt.subplot(8, 8, i + 1)
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.title(class_names[labels_batch[i]])
        plt.axis("off")
```




```
def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.1, test_split=0.1,
    assert (train_split + test_split + val_split) == 1
    ds_size = len(ds)
    if shuffle:
        ds = ds.shuffle(shuffle_size, seed=12)
    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)
    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size)
```

```
    test_ds = ds.skip(train_size).skip(val_size)
    # Autotune all the 3 datasets
    train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUN
    val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
    test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
    return train_ds, val_ds, test_ds


train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)


resize_and_rescale = tf.keras.Sequential([
  layers.experimental.preprocessing.Resizing(IMAGE_SIZE, IMAGE_SIZE),
  layers.experimental.preprocessing.Rescaling(1./255),
])


data_augmentation = tf.keras.Sequential([
  layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
  layers.experimental.preprocessing.RandomRotation(0.2),
])


input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 9

model = models.Sequential([
    resize_and_rescale,
    # data_augmentation,
    layers.Conv2D(32, kernel_size = (3,3), activation='relu', input_shape=input_sh
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='softmax'),
])
model.build(input_shape=input_shape)
```

```
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (62, 256, 256, 3)         0

 conv2d (Conv2D)             (62, 254, 254, 32)        896

 max_pooling2d (MaxPooling2D  (62, 127, 127, 32)       0
 )

 conv2d_1 (Conv2D)           (62, 125, 125, 64)        18496

 max_pooling2d_1 (MaxPooling  (62, 62, 62, 64)         0
 2D)

 conv2d_2 (Conv2D)           (62, 60, 60, 64)          36928

 max_pooling2d_2 (MaxPooling  (62, 30, 30, 64)         0
 2D)

 conv2d_3 (Conv2D)           (62, 28, 28, 64)          36928

 max_pooling2d_3 (MaxPooling  (62, 14, 14, 64)         0
 2D)

 conv2d_4 (Conv2D)           (62, 12, 12, 64)          36928

 max_pooling2d_4 (MaxPooling  (62, 6, 6, 64)           0
 2D)

 conv2d_5 (Conv2D)           (62, 4, 4, 64)            36928

 max_pooling2d_5 (MaxPooling  (62, 2, 2, 64)           0
 2D)

 flatten (Flatten)           (62, 256)                 0

 dense (Dense)               (62, 64)                  16448

 dense_1 (Dense)             (62, 9)                   585

=================================================================
Total params: 184,137
Trainable params: 184,137
Non-trainable params: 0
_________________________________________________________________
```

```
history = model.fit(
    train_ds,
    batch_size=BATCH_SIZE,
    validation_data=val_ds,
    verbose=1,
    epochs=EPOCHS,
)
```

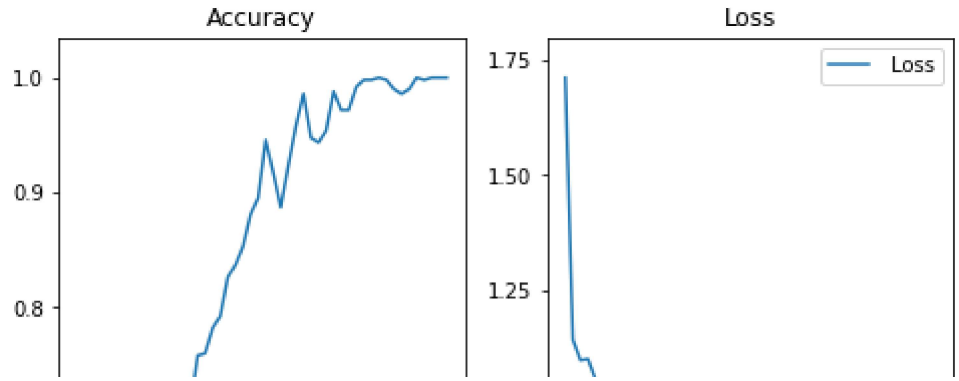
```
Epoch 1/50
8/8 [==============================] - 49s 5s/step - loss: 1.7104 - accuracy: 0.3105
Epoch 2/50
8/8 [==============================] - 37s 5s/step - loss: 1.1452 - accuracy: 0.3548
Epoch 3/50
8/8 [==============================] - 37s 5s/step - loss: 1.1007 - accuracy: 0.3810
Epoch 4/50
8/8 [==============================] - 37s 5s/step - loss: 1.1034 - accuracy: 0.3750
Epoch 5/50
8/8 [==============================] - 37s 5s/step - loss: 1.0588 - accuracy: 0.4194
Epoch 6/50
8/8 [==============================] - 37s 5s/step - loss: 0.9987 - accuracy: 0.4919
Epoch 7/50
8/8 [==============================] - 37s 5s/step - loss: 0.9502 - accuracy: 0.5060
Epoch 8/50
8/8 [==============================] - 37s 5s/step - loss: 0.8960 - accuracy: 0.5887
Epoch 9/50
8/8 [==============================] - 37s 5s/step - loss: 0.8644 - accuracy: 0.5847
Epoch 10/50
8/8 [==============================] - 37s 5s/step - loss: 0.8516 - accuracy: 0.5847
Epoch 11/50
8/8 [==============================] - 37s 5s/step - loss: 0.7975 - accuracy: 0.6512
Epoch 12/50
8/8 [==============================] - 37s 5s/step - loss: 0.7848 - accuracy: 0.6310
Epoch 13/50
8/8 [==============================] - 37s 5s/step - loss: 0.7357 - accuracy: 0.6532
Epoch 14/50
8/8 [==============================] - 37s 5s/step - loss: 0.6856 - accuracy: 0.6754
Epoch 15/50
8/8 [==============================] - 37s 5s/step - loss: 0.6869 - accuracy: 0.6633
Epoch 16/50
8/8 [==============================] - 37s 5s/step - loss: 0.6399 - accuracy: 0.7117
Epoch 17/50
8/8 [==============================] - 37s 5s/step - loss: 0.5815 - accuracy: 0.7581
Epoch 18/50
8/8 [==============================] - 37s 5s/step - loss: 0.5475 - accuracy: 0.7601
Epoch 19/50
8/8 [==============================] - 37s 5s/step - loss: 0.5016 - accuracy: 0.7823
Epoch 20/50
8/8 [==============================] - 37s 5s/step - loss: 0.4751 - accuracy: 0.7923
Epoch 21/50
8/8 [==============================] - 37s 5s/step - loss: 0.3891 - accuracy: 0.8266
Epoch 22/50
8/8 [==============================] - 37s 5s/step - loss: 0.4010 - accuracy: 0.8367
Epoch 23/50
```

```
8/8 [==============================] - 37s 5s/step - loss: 0.3737 - accuracy: 0.8528
Epoch 24/50
8/8 [==============================] - 37s 5s/step - loss: 0.3090 - accuracy: 0.8810
Epoch 25/50
8/8 [==============================] - 37s 5s/step - loss: 0.2650 - accuracy: 0.8952
Epoch 26/50
8/8 [==============================] - 38s 5s/step - loss: 0.1863 - accuracy: 0.9456
Epoch 27/50
8/8 [==============================] - 39s 5s/step - loss: 0.1905 - accuracy: 0.9173
Epoch 28/50
8/8 [==============================] - 38s 5s/step - loss: 0.2624 - accuracy: 0.8871
Epoch 29/50
```

```
model.evaluate(test_ds)
```

```
1/1 [==============================] - 3s 3s/step - loss: 0.3255 - accuracy: 0.9516
[0.325544536113739, 0.9516128897666931]
```

```
acc = history.history['accuracy']
loss = history.history['loss']


plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(range(EPOCHS), acc, label=' Accuracy')
plt.legend(loc='lower right')
plt.title('Accuracy')

plt.subplot(1, 2, 2)
plt.plot(range(EPOCHS), loss, label=' Loss')
plt.legend(loc='upper right')
plt.title('Loss')
plt.show()
```




```
image_path = "/gdrive/My Drive/Multinomial animal classification/dogs/dog.5.jpg"
image = preprocessing.image.load_img(image_path)
image_array = preprocessing.image.img_to_array(image)
scaled_img = np.expand_dims(image_array, axis=0)
image
```

```
pred = model.predict(scaled_img)
```

```
output = class_names[np.argmax(pred)]
```

```
output
```

```
'dogs'
```

```
model.save("Multinomial animal classification.h5")
```