```
# installing the Kaggle library
!pip install kaggle
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (1.5.16)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (from kaggle) (1.16.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from kaggle) (2024.2.2)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.31.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from kaggle) (4.66.2)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle) (8.0.4)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.0.7)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from kaggle) (6.1.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->kaggle) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-packages (from python-slugify->kaggle) (1
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.6)
```

```
# configuring the path of Kaggle.json file
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

Importing the Dog vs Cat Dataset from Kaggle

```
# Kaggle api
!kaggle competitions download -c dogs-vs-cats
```

```
Downloading dogs-vs-cats.zip to /content
100% 810M/812M [00:32<00:00, 32.5MB/s]
100% 812M/812M [00:32<00:00, 26.4MB/s]
```

```
!ls
```

```
dogs-vs-cats.zip  kaggle.json  sample_data
```

```
# extracting the compressed dataset
from zipfile import ZipFile

dataset = '/content/dogs-vs-cats.zip'

with ZipFile(dataset, 'r') as zip:
  zip.extractall()
  print('The dataset is extracted successfully')
```

```
The dataset is extracted successfully
```

```
# extracting the compressed dataset
from zipfile import ZipFile

dataset = '/content/train.zip'

with ZipFile(dataset, 'r') as zip:
  zip.extractall()
  print('The dataset is extracted')
```

```
The dataset is extracted
```

```
import os
# counting the number of files in train folder
path, dirs, files = next(os.walk('/content/train'))
file_count = len(files)
print('Number of images: ', file_count)
```

```
Number of images:  25000
```

Printing the name of images

```
file_names = os.listdir('/content/train/')
print(file_names)
```

```
['dog.6044.jpg', 'dog.6786.jpg', 'cat.11106.jpg', 'cat.11855.jpg', 'cat.1929.jpg', 'dog.5236.jpg', 'dog.7356.jpg', 'cat.6600.j
```

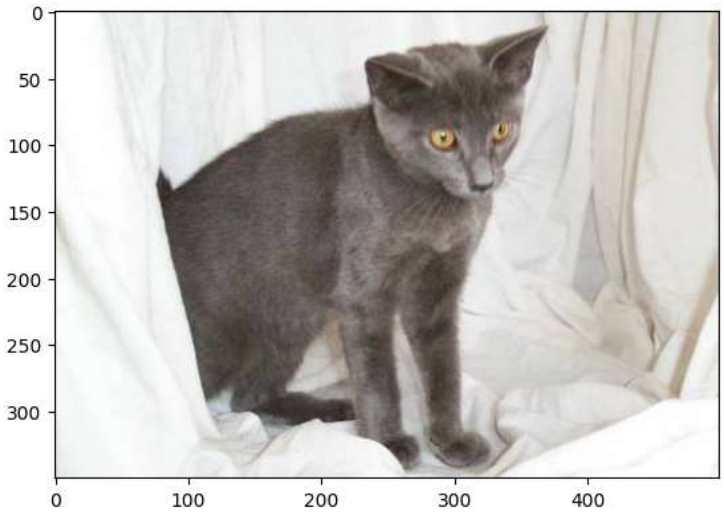
Importing the Dependencies

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from sklearn.model_selection import train_test_split
from google.colab.patches import cv2_imshow
```

Displaying the images of dogs and cats

```
# display dog image
img = mpimg.imread('/content/train/dog.8298.jpg')
imgplt = plt.imshow(img)
plt.show()
```


```
# display cat image
img = mpimg.imread('/content/train/cat.4337.jpg')
imgplt = plt.imshow(img)
plt.show()
```




```
file_names = os.listdir('/content/train/')

for i in range(5):

  name = file_names[i]
  print(name[0:3])
```

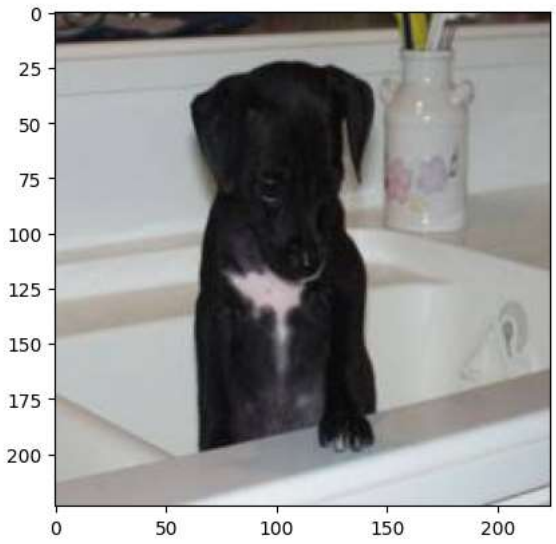
```
dog
dog
cat
cat
cat
```

```
file_names = os.listdir('/content/train/')

dog_count = 0
cat_count = 0

for img_file in file_names:

  name = img_file[0:3]

  if name == 'dog':
    dog_count += 1

  else:
    cat_count += 1

print('Number of dog images =', dog_count)
print('Number of cat images =', cat_count)
```

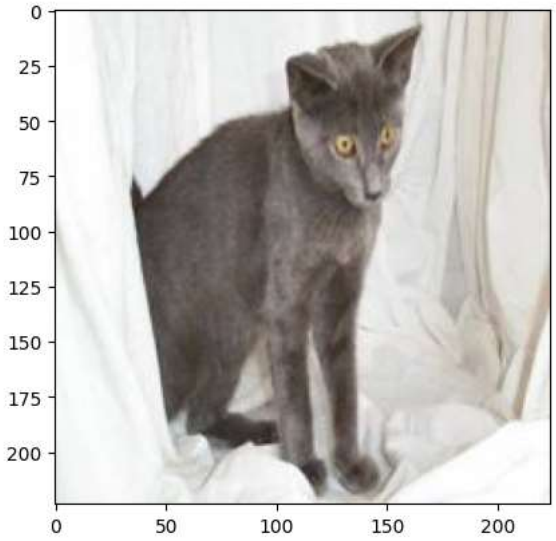
```
Number of dog images = 12500
Number of cat images = 12500
```

Resizing all the images

```
#creating a directory for resized images
os.mkdir('/content/image resized')


original_folder = '/content/train/'
resized_folder = '/content/image resized/'

for i in range(2000):

  filename = os.listdir(original_folder)[i]
  img_path = original_folder+filename

  img = Image.open(img_path)
  img = img.resize((224, 224))
  img = img.convert('RGB')

  newImgPath = resized_folder+filename
  img.save(newImgPath)


# display resized dog image
img = mpimg.imread('/content/image resized/dog.8298.jpg')
imgplt = plt.imshow(img)
plt.show()
```




```
# display resized cat image
img = mpimg.imread ('/content/image resized/cat.4337.jpg')
imgplt = plt.imshow(img)
plt.show()
```



Creating labels for resized images of dogs and cats

Cat –> 0

Dog –> 1

```
# creaing a for loop to assign labels
filenames = os.listdir('/content/image resized/')


labels = []

for i in range(2000):

  file_name = filenames[i]
  label = file_name[0:3]

  if label == 'dog':
    labels.append(1)

  else:
    labels.append(0)


print(filenames[0:5])
print(len(filenames))
```

```
['dog.6044.jpg', 'dog.6786.jpg', 'cat.11106.jpg', 'cat.11855.jpg', 'cat.1929.jpg']
2000
```

```
print(labels[0:5])
print(len(labels))
```

```
[1, 1, 0, 0, 0]
2000
```

```
# counting the images of dogs and cats out of 2000 images
values, counts = np.unique(labels, return_counts=True)
print(values)
print(counts)
```

Converting all the resized images to numpy arrays

```
import cv2
import glob


image_directory = '/content/image resized/'
image_extension = ['png', 'jpg']

files = []

[files.extend(glob.glob(image_directory + '*.' + e)) for e in image_extension]

dog_cat_images = np.asarray([cv2.imread(file) for file in files])
```

```
print(dog_cat_images)
```

```
[[[[ 76  70  75]
   [ 48  42  47]
   [ 42  39  41]
   ...
   [138 143 141]
   [110 116 111]
   [ 97 103  98]]

  [[ 70  64  69]
   [ 52  46  51]
   [ 50  47  49]
   ...
   [128 133 131]
   [114 120 115]
   [113 119 114]]

  [[ 66  60  65]
   [ 58  52  57]
   [ 60  57  59]
   ...
   [122 125 123]
   [128 131 129]
   [139 142 140]]

  ...

  [[ 63  55  55]
   [ 64  56  56]
   [ 68  61  58]
   ...
   [ 74  66  53]
   [ 66  58  45]
   [ 70  62  49]]

  [[ 60  52  52]
   [ 61  53  53]
   [ 68  61  58]
   ...
   [ 57  49  36]
   [ 82  74  61]
   [124 116 103]]

  [[ 64  56  56]
   [ 63  55  55]
   [ 66  59  56]
   ...
   [ 85  77  64]
   [121 113 100]
   [155 147 134]]]


 [[[132  95  69]
   [136  99  73]
   [143 106  80]
   ...
   [ 46  54  71]
   [ 46  53  70]
   [ 46  53  70]]
```

```
type(dog_cat_images)
```

```
numpy.ndarray
```

```
print(dog_cat_images.shape)
```

```
(2000, 224, 224, 3)
```

```
X = dog_cat_images
Y = np.asarray(labels)
```

Train Test Split

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=2)
```

```
print(X.shape, X_train.shape, X_test.shape)
```

```
(2000, 224, 224, 3) (1600, 224, 224, 3) (400, 224, 224, 3)
```

```
# scaling the data
X_train_scaled = X_train/255

X_test_scaled = X_test/255


print(X_train_scaled)
```

```
[[[[0.68627451 0.72156863 0.87058824]
   [0.65882353 0.69411765 0.84313725]
   [0.60392157 0.63921569 0.78823529]
   ...
   [0.57647059 0.69411765 0.7372549 ]
   [0.58039216 0.69803922 0.74117647]
   [0.58039216 0.69803922 0.74117647]]

  [[0.64313725 0.67843137 0.82745098]
   [0.60392157 0.63921569 0.78823529]
   [0.56078431 0.59607843 0.74509804]
   ...
   [0.59215686 0.70980392 0.75294118]
   [0.59215686 0.70980392 0.75294118]
   [0.59607843 0.71372549 0.75686275]]

  [[0.59607843 0.63137255 0.78039216]
   [0.57254902 0.60784314 0.75686275]
   [0.54509804 0.58039216 0.72941176]
   ...
   [0.6        0.71764706 0.76078431]
   [0.6        0.71764706 0.76078431]
   [0.6        0.71764706 0.76078431]]

  ...

  [[0.23529412 0.16078431 0.12941176]
   [0.24313725 0.16862745 0.1372549 ]
   [0.24705882 0.17254902 0.14117647]
   ...
   [0.45098039 0.55686275 0.57254902]
   [0.46666667 0.58039216 0.59607843]
   [0.48627451 0.60784314 0.61960784]]

  [[0.24705882 0.17254902 0.14117647]
   [0.24705882 0.17254902 0.14117647]
   [0.25098039 0.17647059 0.14509804]
   ...
   [0.45098039 0.55686275 0.57254902]
   [0.47058824 0.58431373 0.6       ]
   [0.49411765 0.61568627 0.62745098]]

  [[0.25098039 0.17647059 0.14509804]
   [0.25098039 0.17647059 0.14509804]
   [0.24705882 0.17254902 0.14117647]
   ...
   [0.44313725 0.55686275 0.57254902]
   [0.4745098  0.58823529 0.60392157]
   [0.49803922 0.61960784 0.63137255]]]


 [[[0.         0.31764706 0.46666667]
   [0.         0.29411765 0.44313725]
   [0.         0.2745098  0.41960784]
   ...
   [0.23137255 0.77254902 0.8627451 ]
   [0.18039216 0.78823529 0.87058824]
   [0.14117647 0.78823529 0.8627451 ]]
```

Building the Neural Network

```
import tensorflow as tf
import tensorflow_hub as hub


mobilenet_model = 'https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4'

pretrained_model = hub.KerasLayer(mobilenet_model, input_shape=(224,224,3), trainable=False)
```

```
num_of_classes = 2

model = tf.keras.Sequential([

    pretrained_model,
    tf.keras.layers.Dense(num_of_classes)

])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 keras_layer (KerasLayer)    (None, 1280)              2257984

 dense (Dense)               (None, 2)                 2562

=================================================================
Total params: 2260546 (8.62 MB)
Trainable params: 2562 (10.01 KB)
Non-trainable params: 2257984 (8.61 MB)
_________________________________________________________________
```

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


model.fit(X_train_scaled, Y_train, epochs=5)
```

```
Epoch 1/5
50/50 [==============================] - 77s 1s/step - loss: 0.1693 - acc: 0.9344
Epoch 2/5
50/50 [==============================] - 66s 1s/step - loss: 0.0679 - acc: 0.9781
Epoch 3/5
50/50 [==============================] - 70s 1s/step - loss: 0.0514 - acc: 0.9862
Epoch 4/5
50/50 [==============================] - 66s 1s/step - loss: 0.0410 - acc: 0.9881
Epoch 5/5
50/50 [==============================] - 76s 2s/step - loss: 0.0340 - acc: 0.9906
<keras.src.callbacks.History at 0x7aea302c1e70>
```

```
score, acc = model.evaluate(X_test_scaled, Y_test)
print('Test Loss =', score)
print('Test Accuracy =', acc)
```

```
13/13 [==============================] - 18s 1s/step - loss: 0.1067 - acc: 0.9650
Test Loss = 0.10671394318342209
Test Accuracy = 0.9649999737739563
```

Predictive System

```
input_image_path = input('Path of the image to be predicted: ')

input_image = cv2.imread(input_image_path)

cv2_imshow(input_image)

input_image_resize = cv2.resize(input_image, (224,224))

input_image_scaled = input_image_resize/255

image_reshaped = np.reshape(input_image_scaled, [1,224,224,3])

input_prediction = model.predict(image_reshaped)

print(input_prediction)

input_pred_label = np.argmax(input_prediction)

print(input_pred_label)

if input_pred_label == 0:
  print('The image represents a Cat')

else:
  print('The image represents a Dog')
```

```
Path of the image to be predicted: /content/image resized/cat.10061.jpg
```

```
1/1 [==============================] - 1s 606ms/step
[[ 2.4227407 -2.5215437]]
0
The image represents a Cat
```

```
input_image_path = input('Path of the image to be predicted: ')

input_image = cv2.imread(input_image_path)

cv2_imshow(input_image)

input_image_resize = cv2.resize(input_image, (224,224))

input_image_scaled = input_image_resize/255

image_reshaped = np.reshape(input_image_scaled, [1,224,224,3])

input_prediction = model.predict(image_reshaped)

print(input_prediction)

input_pred_label = np.argmax(input_prediction)

print(input_pred_label)

if input_pred_label == 0:
  print('The image represents a Cat')

else:
  print('The image represents a Dog')
```

```
Path of the image to be predicted: /content/image resized/dog.10094.jpg
```

```
1/1 [==============================] - 0s 52ms/step
[[-3.0304697  2.6972792]]
1
The image represents a Dog
```

+ Code + Text