DOG VS CAT IMAGE CLASSIFICATION

Dog vs. cat image classification in Python involves using machine learning libraries such as TensorFlow or PyTorch to build a convolutional neural network (CNN) that can distinguish between images of dogs and cats. The process typically includes loading and preprocessing the image dataset, defining the CNN architecture, training the model, and evaluating its performance on a test set.

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

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (1.6.12)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (from kaggle) (1.16.0)
Requirement already satisfied: certifi>=2023.7.22 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2024.2
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) (4.66.4)
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
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests-
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.7
```

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

importing the dataset

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

```
dogs-vs-cats.zip: Skipping, found more recently modified local copy (use --force to force download)
```

```
!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')
```

```
The dataset is extracted
```

```
# 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
```

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

```
['dog.7711.jpg', 'cat.1664.jpg', 'cat.10373.jpg', 'dog.8542.jpg', 'dog.3238.jpg', 'cat.11990.jpg', 'dog.7337.jpg',
```

importing the libraries

```
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
```

import numpy as np: Used for numerical operations on arrays, which are essential for handling image data.

from PIL import Image: Allows for opening, manipulating, and saving image files in various formats.

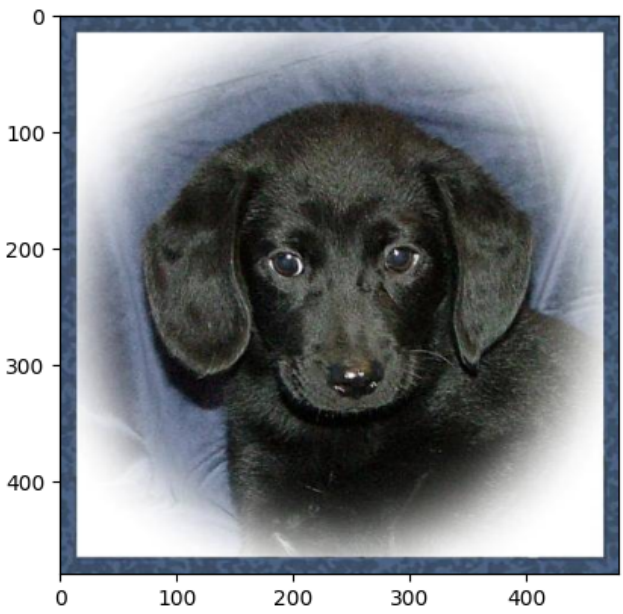
import matplotlib.pyplot as plt: Provides tools for plotting and visualizing data, including images.

import matplotlib.image as mpimg: Used to read images into arrays for processing and visualization.

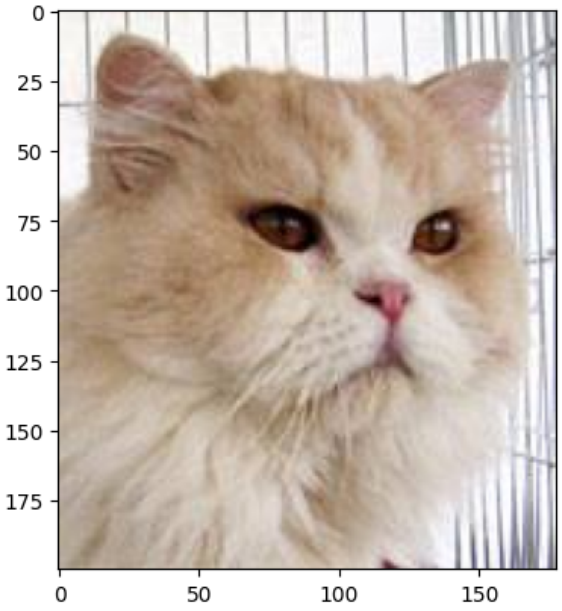
from sklearn.model_selection import train_test_split: Facilitates splitting the dataset into training and testing sets to evaluate the performance of a model.

from google.colab.patches import cv2_imshow: Allows displaying images in Google Colab using OpenCV, which is not natively supported.

```
# display dog image
img = mpimg.imread('/content/train/dog.7711.jpg')
imgplt = plt.imshow(img)
plt.show()
```



```
# display cat image
img = mpimg.imread('/content/train/cat.1275.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
cat
cat
dog
dog
```

The code lists the filenames in the '/content/train/' directory, then iterates through the first five filenames, printing the first three characters of each.

```
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
```

The code counts the number of dog and cat images in the '/content/train/' directory by checking if the first three characters of each filename are 'dog' or not, and then prints the respective counts.

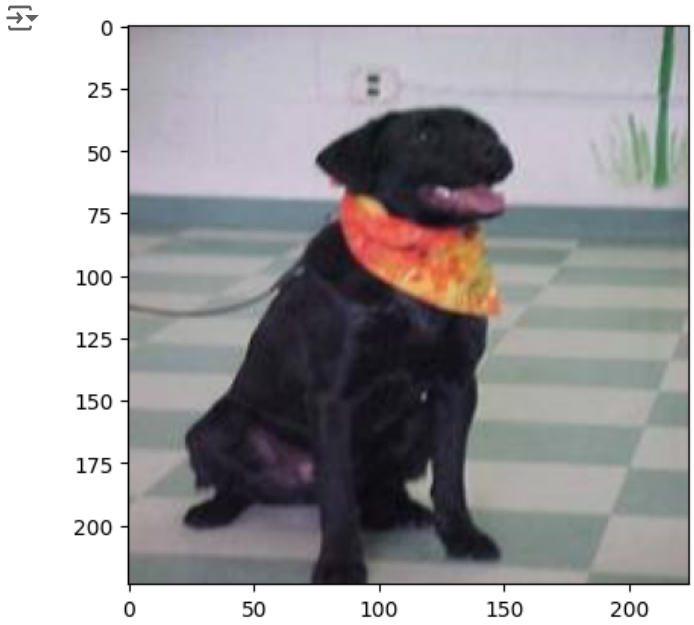
Resizing all the images

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

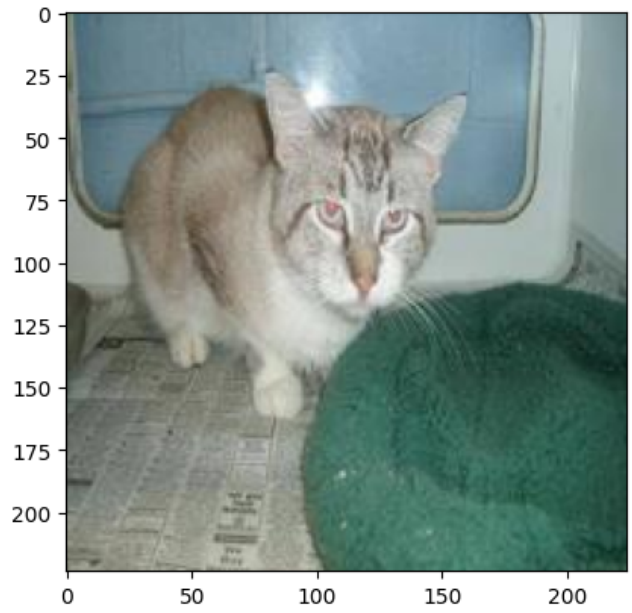
```
original_folder = '/content/train/'
resized_folder = '/content/image resized/'

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)
```

This code resizes the first 2000 images in the '/content/train/' directory to 224x224 pixels, converts them to RGB format, and saves the resized images to the '/content/image resized/' directory with the same filenames.

```
# display resized dog image
img = mpimg.imread('/content/image resized/dog.10153.jpg')
imgplt = plt.imshow(img)
plt.show()
```




```
# display resized cat image
img = mpimg.imread('/content/image resized/cat.8606.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.7711.jpg', 'cat.1664.jpg', 'cat.10373.jpg', 'dog.8542.jpg', 'dog.3238.jpg']
2000
```

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

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

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

```
[0 1]
[ 984 1016]
```

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)
```

```
[[ 21  79 101]
  [ 19  77  99]
  [ 17  75  97]
  ...
  [  6   8   8]
  [  7   9   9]
  [  7   9   9]]]


 [[[109 109  91]
  [173 174 154]
  [125 128 106]
  ...
  [111 107 106]
  [ 99  95  94]
  [ 93  89  88]]

 [[112 112  94]
  [174 175 155]
  [128 131 109]
  ...
  [118 114 113]
  [100  96  95]
  [ 92  88  87]]

 [[114 114  96]
  [173 174 154]
  [132 135 113]
  ...
  [123 119 118]
  [107 103 102]
  [100  96  95]]

 ...

 [[ 12  10   9]
  [ 14  12  11]
  [ 14  12  11]
  ...
  [ 13  11  10]
  [ 13  11  10]
  [ 13  11  10]]

 [[ 14  12  11]
  [ 19  17  16]
  [ 18  16  15]
  ...
  [ 13  11  10]
  [ 13  11  10]
  [ 13  11  10]]

 [[ 13  11  10]
  [ 21  19  18]
  [ 20  18  17]
  ...
  [ 13  11  10]
  [ 13  11  10]
  [ 13  11  10]]]]
```

```
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)
```

1600 --> training images

400 --> test images

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

X_test_scaled = X_test/255
```

```
print(X_train_scaled)
```

```
[[0.70588235 0.7254902  0.72156863]
 [0.72941176 0.74901961 0.74509804]
 [0.71764706 0.7372549  0.73333333]
 ...
 [0.67058824 0.69411765 0.6745098 ]
 [0.6627451  0.68627451 0.66666667]
 [0.64705882 0.67058824 0.65098039]]

[[0.70588235 0.7254902  0.72156863]
 [0.7254902  0.74509804 0.74117647]
 [0.70980392 0.72941176 0.7254902 ]
 ...
 [0.66666667 0.69019608 0.67058824]
 [0.66666667 0.69019608 0.67058824]
 [0.64313725 0.66666667 0.64705882]]]]
```

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 [==============================] - 63s 1s/step - loss: 0.1535 - acc: 0.9488
Epoch 2/5
50/50 [==============================] - 54s 1s/step - loss: 0.0635 - acc: 0.9800
Epoch 3/5
50/50 [==============================] - 59s 1s/step - loss: 0.0449 - acc: 0.9881
Epoch 4/5
50/50 [==============================] - 56s 1s/step - loss: 0.0361 - acc: 0.9900
Epoch 5/5
50/50 [==============================] - 57s 1s/step - loss: 0.0276 - acc: 0.9937
<keras.src.callbacks.History at 0x7fa8690af400>
```

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

```
13/13 [==============================] - 15s 1s/step - loss: 0.0648 - acc: 0.9800
Test Loss = 0.06484432518482208
Test Accuracy = 0.9800000190734863
```

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/cat.jpeg
```

```
1/1 [==============================] - 0s 476ms/step
[[ 3.9422524 -4.4835076]]
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/dog.jpg
```

```
1/1 [==============================] - 0s 82ms/step
[[-4.565148   3.5642705]]
1
The image represents a Dog
```