Cat Vs Dog Classification

Google A
Arindal C
arindalch

Using Transfer Learning

Installing 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) (2023.7.22)
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) (4.66.1)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle) (8.0.1)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.0.4)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from kaggle) (6.0.0)
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) (1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.2.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.4)
```

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 competitions download -c dogs-vs-cats
```

```
Downloading dogs-vs-cats.zip to /content
 98% 793M/812M [00:06<00:00, 178MB/s]
100% 812M/812M [00:06<00:00, 132MB/s]
```

```
!ls
```

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

Extracting the Zip file

```
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
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 the Images

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

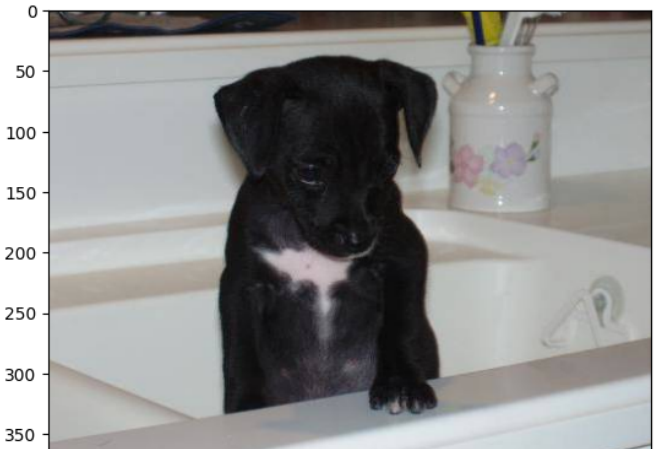
```
['cat.1172.jpg', 'cat.10592.jpg', 'cat.5239.jpg', 'dog.2769.jpg', 'dog.11306.jpg', 'cat.5669.jpg', 'dog.7115.jpg', 'dog.4090.jpg',
```

Importing 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 and Image of a Cat and a Dog

```
# 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.4352.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])
```

```
cat
cat
cat
dog
dog
```

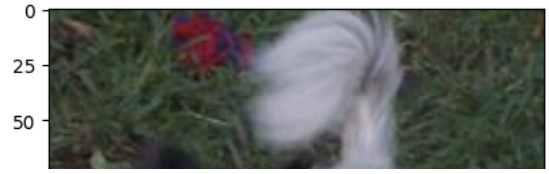
Counting number of Cats and Dogs Images

```
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)
```

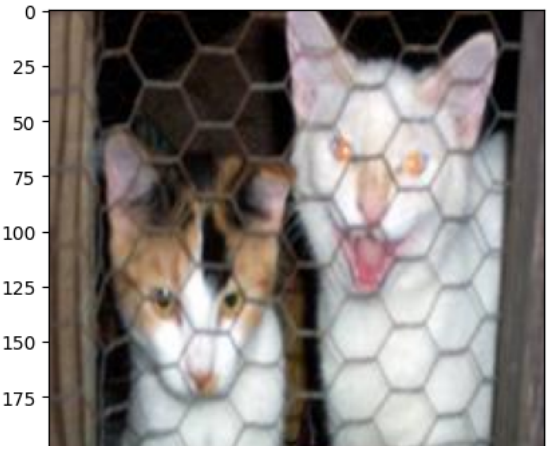
```
Number of dog images = 12500
Number of cat images = 12500
```

Resizing all 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.10012.jpg')
imgplt = plt.imshow(img)
plt.show()
```




```
# display resized cat image
img = mpimg.imread('/content/image resized/cat.9351.jpg')
imgplt = plt.imshow(img)
plt.show()
```



Creating a Label for the Classification

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))
```

```
['cat.1172.jpg', 'cat.10592.jpg', 'cat.5239.jpg', 'dog.2769.jpg', 'dog.11306.jpg']
2000
```

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

```
[0, 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]
[ 991 1009]
```

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)

type(dog_cat_images)
```

```
        [ 40  45  48]
        ...
        [ 41  58  67]
        [ 41  58  67]
        [ 41  58  67]]]


      [[[ 40  49  58]
        [ 33  44  52]
        [ 60  71  79]
        ...
        [154 169 165]
        [238 253 249]
        [232 249 245]]

       [[ 51  60  69]
        [ 61  72  80]
        [ 96 109 117]
        ...
        [157 171 167]
        [226 241 237]
        [239 255 252]]

       [[ 92 103 111]
        [102 113 121]
        [124 137 145]
        ...
        [161 174 172]
        [221 235 233]
        [243 255 255]]

       ...

       [[ 85 112 133]
        [ 98 125 146]
        [139 166 187]
        ...
        [ 11  10  26]
        [ 18  17  33]
        [ 24  23  39]]

       [[100 127 148]
        [ 93 120 141]
        [ 85 112 133]
        ...
        [  6   4  23]
        [ 11   9  28]
        [ 16  14  33]]

       [[106 133 154]
        [113 140 161]
        [128 155 176]
        ...
        [  0   0  15]
        [  0   0  17]
        [  1   0  19]]]


      [[[128 139 143]
```

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

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

Assigning the values to Labels

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

Spliting the Dataset

```
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)
```

```
        [0.14509804 0.19215686 0.20784314]
        [0.21176471 0.25098039 0.27843137]
        ...
        [0.14901961 0.55294118 0.49411765]
        [0.17254902 0.61568627 0.48627451]
        [0.30588235 0.78039216 0.60784314]]]


      [[[1.         1.         1.        ]
        [1.         1.         1.        ]
        [1.         1.         1.        ]
        ...
        [1.         1.         1.        ]
        [1.         1.         1.        ]
        [1.         1.         1.        ]]

       [[1.         1.         1.        ]
        [1.         1.         1.        ]
        [1.         1.         1.        ]
        ...
        [1.         1.         1.        ]
        [1.         1.         1.        ]
        [1.         1.         1.        ]]

       [[1.         1.         1.        ]
        [1.         1.         1.        ]
        [1.         1.         1.        ]
        ...
        [1.         1.         1.        ]
        [1.         1.         1.        ]
        [1.         1.         1.        ]]

       ...

       [[1.         1.         1.        ]
        [1.         1.         1.        ]
        [1.         1.         1.        ]
        ...
        [1.         1.         1.        ]
        [1.         1.         1.        ]
        [1.         1.         1.        ]]

       [[1.         1.         1.        ]
        [1.         1.         1.        ]
        [1.         1.         1.        ]
        ...
        [1.         1.         1.        ]
        [1.         1.         1.        ]
        [1.         1.         1.        ]]

       [[1.         1.         1.        ]
        [1.         1.         1.        ]
        [1.         1.         1.        ]
        ...
        [1.         1.         1.        ]
        [1.         1.         1.        ]
        [1.         1.         1.        ]]]
```

Building the Neural Network

Double-click (or enter) to edit

```
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: 2,260,546
Trainable params: 2,562
Non-trainable params: 2,257,984
_________________________________________________________________
```

```
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 [==============================] - 70s 1s/step - loss: 0.2176 - acc: 0.9050
Epoch 2/5
50/50 [==============================] - 72s 1s/step - loss: 0.0668 - acc: 0.9769
Epoch 3/5
50/50 [==============================] - 65s 1s/step - loss: 0.0475 - acc: 0.9862
Epoch 4/5
50/50 [==============================] - 66s 1s/step - loss: 0.0387 - acc: 0.9894
Epoch 5/5
50/50 [==============================] - 66s 1s/step - loss: 0.0315 - acc: 0.9919
<keras.callbacks.History at 0x7e369c244df0>
```

```
score, acc = model.evaluate(X_test_scaled, Y_test)
print('Test Loss =', score)
print('Test Accuracy =', acc)
print(acc*100)
```

```
13/13 [==============================] - 21s 2s/step - loss: 0.0624 - acc: 0.9750
Test Loss = 0.062394656240940094
Test Accuracy = 0.9750000238418579
97.50000238418579
```

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/cat2.jpg
```

```
1/1 [==============================] - 0s 102ms/step
[[ 4.105035 -4.032446]]
0
The image represents a Cat
```