Dog Breed Prediction

In this project, we will see how to use Keras and TensorFlow to build, train, and test a Convolutional Neural Network capable of identifying the breed of a dog in a supplied image. This is a supervised learning problem, specifically a multiclass classification problem.

```
print("Hello")
```

```
Hello
```

```
pip install google.colab
```

```
Requirement already satisfied: traitlets>=4.1.0 in /usr/local/lib/python3.8/dist-packages (from ipykernel~=5.3.4->google.colab)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.8/dist-packages (from ipython~=7.9.0->google.colab) (0.7.5)
Requirement already satisfied: prompt-toolkit<2.1.0,>=2.0.0 in /usr/local/lib/python3.8/dist-packages (from ipython~=7.9.0->goog
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.8/dist-packages (from ipython~=7.9.0->google.colab) (5
Requirement already satisfied: backcall in /usr/local/lib/python3.8/dist-packages (from ipython~=7.9.0->google.colab) (0.2.0)
Requirement already satisfied: decorator in /usr/local/lib/python3.8/dist-packages (from ipython~=7.9.0->google.colab) (4.4.2)
Requirement already satisfied: pygments in /usr/local/lib/python3.8/dist-packages (from ipython~=7.9.0->google.colab) (2.6.1)
Requirement already satisfied: pexpect in /usr/local/lib/python3.8/dist-packages (from ipython~=7.9.0->google.colab) (4.8.0)
Collecting jedi>=0.10
  Downloading jedi-0.18.2-py2.py3-none-any.whl (1.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 19.4 MB/s eta 0:00:00
Requirement already satisfied: argon2-cffi in /usr/local/lib/python3.8/dist-packages (from notebook~=6.3.0->google.colab) (21.3.
Requirement already satisfied: pyzmq>=17 in /usr/local/lib/python3.8/dist-packages (from notebook~=6.3.0->google.colab) (23.2.1)
Requirement already satisfied: prometheus-client in /usr/local/lib/python3.8/dist-packages (from notebook~=6.3.0->google.colab)
Requirement already satisfied: nbformat in /usr/local/lib/python3.8/dist-packages (from notebook~=6.3.0->google.colab) (5.7.3)
Requirement already satisfied: terminado>=0.8.3 in /usr/local/lib/python3.8/dist-packages (from notebook~=6.3.0->google.colab) (
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.8/dist-packages (from notebook~=6.3.0->google.colab) (
Requirement already satisfied: nbconvert in /usr/local/lib/python3.8/dist-packages (from notebook~=6.3.0->google.colab) (6.5.4)
Requirement already satisfied: jupyter-core>=4.6.1 in /usr/local/lib/python3.8/dist-packages (from notebook~=6.3.0->google.colab
Requirement already satisfied: jinja2 in /usr/local/lib/python3.8/dist-packages (from notebook~=6.3.0->google.colab) (3.1.2)
Requirement already satisfied: Send2Trash>=1.5.0 in /usr/local/lib/python3.8/dist-packages (from notebook~=6.3.0->google.colab)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-packages (from pandas>=1.1.0->google.colab) (2022.7
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.8/dist-packages (from pandas>=1.1.0->google.cola
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.8/dist-packages (from pandas>=1.1.0->google.colab) (1.24.
Requirement already satisfied: chardet<5,>=3.0.2 in /usr/local/lib/python3.8/dist-packages (from requests>=2.25.1->google.colab)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.8/dist-packages (from requests>=2.25.1->google.co
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-packages (from requests>=2.25.1->google.colab) (2.1
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/dist-packages (from requests>=2.25.1->google.colab
Requirement already satisfied: parso<0.9.0,>=0.8.0 in /usr/local/lib/python3.8/dist-packages (from jedi>=0.10->ipython~=7.9.0->g
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.8/dist-packages (from jupyter-core>=4.6.1->notebook~=
Requirement already satisfied: wcwidth in /usr/local/lib/python3.8/dist-packages (from prompt-toolkit<2.1.0,>=2.0.0->ipython~=7.
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.8/dist-packages (from pyasn1-modules>=0.2.1->googl
Requirement already satisfied: ptyprocess in /usr/local/lib/python3.8/dist-packages (from terminado>=0.8.3->notebook~=6.3.0->goo
Requirement already satisfied: argon2-cffi-bindings in /usr/local/lib/python3.8/dist-packages (from argon2-cffi->notebook~=6.3.0
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.8/dist-packages (from jinja2->notebook~=6.3.0->google.c
Requirement already satisfied: tinycss2 in /usr/local/lib/python3.8/dist-packages (from nbconvert->notebook~=6.3.0->google.colab
Requirement already satisfied: lxml in /usr/local/lib/python3.8/dist-packages (from nbconvert->notebook~=6.3.0->google.colab) (4
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.8/dist-packages (from nbconvert->notebook~=6.3.0->googl
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python3.8/dist-packages (from nbconvert->notebook~=6.3.0->goo
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.8/dist-packages (from nbconvert->notebook~=6.3.0->google
Requirement already satisfied: packaging in /usr/local/lib/python3.8/dist-packages (from nbconvert->notebook~=6.3.0->google.cola
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.8/dist-packages (from nbconvert->notebook~=6.3.0->g
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.8/dist-packages (from nbconvert->notebook~=6.3.0->
Requirement already satisfied: bleach in /usr/local/lib/python3.8/dist-packages (from nbconvert->notebook~=6.3.0->google.colab)
Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/lib/python3.8/dist-packages (from nbconvert->notebook~=6.3.0->go
Requirement already satisfied: defusedxml in /usr/local/lib/python3.8/dist-packages (from nbconvert->notebook~=6.3.0->google.col
Requirement already satisfied: fastjsonschema in /usr/local/lib/python3.8/dist-packages (from nbformat->notebook~=6.3.0->google.
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.8/dist-packages (from nbformat->notebook~=6.3.0->google
Requirement already satisfied: pyrsistent!=0.17.0,!=0.17.1,!=0.17.2,>=0.14.0 in /usr/local/lib/python3.8/dist-packages (from jso
Requirement already satisfied: attrs>=17.4.0 in /usr/local/lib/python3.8/dist-packages (from jsonschema>=2.6->nbformat->notebook
Requirement already satisfied: importlib-resources>=1.4.0 in /usr/local/lib/python3.8/dist-packages (from jsonschema>=2.6->nbfor
Requirement already satisfied: cffi>=1.0.1 in /usr/local/lib/python3.8/dist-packages (from argon2-cffi-bindings->argon2-cffi->no
Requirement already satisfied: webencodings in /usr/local/lib/python3.8/dist-packages (from bleach->nbconvert->notebook~=6.3.0->
Requirement already satisfied: pycparser in /usr/local/lib/python3.8/dist-packages (from cffi>=1.0.1->argon2-cffi-bindings->argo
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.8/dist-packages (from importlib-resources>=1.4.0->jsonschem
Installing collected packages: jedi
Successfully installed jedi-0.18.2
```

```
pip install Cython
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: Cython in /usr/local/lib/python3.8/dist-packages (0.29.33)
```

```
# Run this cell and select the kaggle.json file downloaded
# from the Kaggle account settings page.
from google.colab import files
files.upload()
```

```
Choose files kaggle.json
• kaggle.json(application/json) - 67 bytes, last modified: 07/03/2023 - 100% done
Saving kaggle.json to kaggle.json
```

We will start by connecting to Kaggle using Kaggle API which can be downloaded from your Kaggle account's settings and uploading it here(upload box).

```
# Next, install the Kaggle API client.
!pip install -q kaggle
```

Next we will install Kaggle API using pip installation.

```
# The Kaggle API client expects this file to be in ~/.kaggle, so move it there.
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/

# This permissions change avoids a warning on Kaggle tool startup.
!chmod 600 ~/.kaggle/kaggle.json
```

Setting up Kaggle using Kaggle API.

```
# Creating directory and changing the current working directory
!mkdir dog_dataset
%cd dog_dataset
```

```
/content/dog_dataset
```

To store the data we will create a new directory and make it as current working directory.

```
# Searching for dataset
!kaggle datasets list -s dogbreedidfromcomp
```

```
Warning: Looks like you're using an outdated API Version, please consider updating (server 1.5.13 / client 1.5.12)
ref                                title                    size  lastUpdated          downloadCount  voteCount  usabilityRating
---------------------------------  ----------------------  -----  -------------------  -------------  ---------  ---------------
catherinehorng/dogbreedidfromcomp  dog-breed-id-from-comp  691MB  2020-06-26 03:09:05           2981          6  0.1764706
```

Searching Kaggle for the required dataset using search option(-s) with title 'dogbreedidfromcomp'. We can also use different search options like searching competitions, notebooks, kernels, datasets, etc.

```
# Downloading dataset and coming out of directory
!kaggle datasets download catherinehorng/dogbreedidfromcomp
%cd ..
```

```
Downloading dogbreedidfromcomp.zip to /content/dog_dataset
 98% 678M/691M [00:05<00:00, 135MB/s]
100% 691M/691M [00:05<00:00, 125MB/s]
/content
```

After searching the data next step would be downloading the data into collab notebook using references found in search option.

```
# Unzipping downloaded file and removing unusable file
!unzip dog_dataset/dogbreedidfromcomp.zip -d dog_dataset
!rm dog_dataset/dogbreedidfromcomp.zip
!rm dog_dataset/sample_submission.csv
```

```
    inflating: dog_dataset/train/ff0931b1c82289dc2cf02f0b4a165139.jpg
    inflating: dog_dataset/train/ff0c4e0e856f1eddcc61facca64440c9.jpg
    inflating: dog_dataset/train/ff0d0773ee3eeb6eb90a172d6afd1ea1.jpg
    inflating: dog_dataset/train/ff0def9dafea6e633d0d7249554fcb2c.jpg
    inflating: dog_dataset/train/ff12508818823987d04e8fa4f5907efe.jpg
    inflating: dog_dataset/train/ff181f0d69202b0650e6e5d76e9c13cc.jpg
    inflating: dog_dataset/train/ff2523c07da7a6cbeeb7c8f8dafed24f.jpg
    inflating: dog_dataset/train/ff3b935868afb51b2d0b75ddc989d058.jpg
    inflating: dog_dataset/train/ff47baef46c5876eaf9a403cd6a54d72.jpg
    inflating: dog_dataset/train/ff4afeb51a1473f7ba18669a8ff48bc9.jpg
    inflating: dog_dataset/train/ff4bb57ce419cd637dd511a1b5474bff.jpg
    inflating: dog_dataset/train/ff52a3909f5801a71161cec95d213107.jpg
    inflating: dog_dataset/train/ff54d45962b3123bb67052e8e29a60e7.jpg
    inflating: dog_dataset/train/ff63ed894f068da8e2bbdfda50a9a9f8.jpg
    inflating: dog_dataset/train/ff63fa05a58473138848f80840064d23.jpg
    inflating: dog_dataset/train/ff6f47aa8e181b6efa4d0be7b09b5628.jpg
    inflating: dog_dataset/train/ff7334b06cee8667a7f30eb00e0b93cf.jpg
    inflating: dog_dataset/train/ff7d9c08091acc3b18b869951feeb013.jpg
    inflating: dog_dataset/train/ff84992beff3edd99b72718bec9448d2.jpg
    inflating: dog_dataset/train/ff8e3fa7e04faca99af85195507ee54d.jpg
    inflating: dog_dataset/train/ff91c3c095a50d3d7f1ab52b60e93638.jpg
    inflating: dog_dataset/train/ffa0055ec324829882186bae29491645.jpg
    inflating: dog_dataset/train/ffa0ad682c6670db3defce2575a2587f.jpg
    inflating: dog_dataset/train/ffa16727a9ee462ee3f386be865b199e.jpg
    inflating: dog_dataset/train/ffa4e1bf959425bad9228b04af40ac76.jpg
    inflating: dog_dataset/train/ffa6a8d29ce57eb760d0f182abada4bf.jpg
    inflating: dog_dataset/train/ffbbf7536ba86dcef3f360bda41181b4.jpg
    inflating: dog_dataset/train/ffc1717fc5b5f7a6c76d0e4ea7c8f93a.jpg
    inflating: dog_dataset/train/ffc2b6b9133a6413c4a013cff29f9ed2.jpg
    inflating: dog_dataset/train/ffc532991d3cd7880d27a449ed1c4770.jpg
    inflating: dog_dataset/train/ffca1c97cea5fada05b8646998a5b788.jpg
    inflating: dog_dataset/train/ffcb610e811817766085054616551f9c.jpg
    inflating: dog_dataset/train/ffcde16e7da0872c357fbc7e2168c05f.jpg
    inflating: dog_dataset/train/ffcffab7e4beef9a9b8076ef2ca51909.jpg
    inflating: dog_dataset/train/ffd25009d635cfd16e793503ac5edef0.jpg
    inflating: dog_dataset/train/ffd3f636f7f379c51ba3648a9ff8254f.jpg
    inflating: dog_dataset/train/ffe2ca6c940cddfee68fa3cc6c63213f.jpg
    inflating: dog_dataset/train/ffe5f6d8e2bff356e9482a80a6e29aac.jpg
    inflating: dog_dataset/train/fff43b07992508bc822f33d8ffd902ae.jpg
```

We will unzip the data which is downloaded and remove the irrelevant files.

```
# Important library imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from tqdm import tqdm
import keras.utils as image
from sklearn.preprocessing import label_binarize
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras.optimizers import Adam
```

Importing required libraries.

```
# Read the labels.csv file and checking shape and records
labels_all = pd.read_csv("dog_dataset/labels.csv")
print(labels_all.shape)
labels_all.head()
```

```
(10222, 2)
```

	id	breed
0	000bec180eb18c7604dcecc8fe0dba07	boston_bull
1	001513dfcb2ffafc82cccf4d8bbaba97	dingo
2	001cdf01b096e06d78e9e5112d419397	pekinese
3	00214f311d5d2247d5dfe4fe24b2303d	bluetick
4	0021f9ceb3235effd7fcde7f7538ed62	golden_retriever

Loading the labels data into dataframe and viewing it. Here we analysed that labels contains 10222 rows and 2 columns.

```
# Visualize the number of each breeds
breeds_all = labels_all["breed"]
breed_counts = breeds_all.value_counts()
breed_counts.head()
```

```
scottish_deerhound      126
maltese_dog             117
afghan_hound            116
entlebucher             115
bernese_mountain_dog    114
Name: breed, dtype: int64
```

Here we are finding out the count per class i.e. total data in each class using value_counts() function.

```
# Selecting first 3 breeds (Limitation due to computation power)
CLASS_NAMES = ['scottish_deerhound','maltese_dog','bernese_mountain_dog']
labels = labels_all[(labels_all['breed'].isin(CLASS_NAMES))]
labels = labels.reset_index()
labels.head()
```

	index	id	breed
0	9	0042188c895a2f14ef64a918ed9c7b64	scottish_deerhound
1	12	00693b8bc2470375cc744a6391d397ec	maltese_dog
2	79	01e787576c003930f96c966f9c3e1d44	scottish_deerhound
3	90	022b34fd8734b39995a9f38a4f3e7b6b	maltese_dog
4	118	02d54f0dfb40038765e838459ae8c956	bernese_mountain_dog

We will work on only 3 breeds due to limited computational power. You can consider more classes as per your system computational power.

```
# Creating numpy matrix with zeros
X_data = np.zeros((len(labels), 224, 224, 3), dtype='float32')
# One hot encoding
Y_data = label_binarize(labels['breed'], classes = CLASS_NAMES)

# Reading and converting image to numpy array and normalizing dataset
for i in tqdm(range(len(labels))):
    img = image.load_img('dog_dataset/train/%s.jpg' % labels['id'][i], target_size=(224, 224))
    img = image.img_to_array(img)
    x = np.expand_dims(img.copy(), axis=0)
    X_data[i] = x / 255.0                #Normalizing Datset

# Printing train image and one hot encode shape & size
print('\nTrain Images shape: ',X_data.shape,' size: {:,}'.format(X_data.size))
print('One-hot encoded output shape: ',Y_data.shape,' size: {:,}'.format(Y_data.size))
```

```
100%|██████████| 357/357 [00:04<00:00, 85.41it/s]
Train Images shape:  (357, 224, 224, 3)  size: 53,738,496
One-hot encoded output shape:  (357, 3)  size: 1,071
```

As we are working with the classification dataset first we need to one hot encode the target value i.e. the classes. After that we will read images and convert them into numpy array and finally normalizing the array.

```
# Building the Model
model = Sequential()

model.add(Conv2D(filters = 64, kernel_size = (5,5), activation ='relu', input_shape = (224,224,3)))
model.add(MaxPool2D(pool_size=(2,2)))

model.add(Conv2D(filters = 32, kernel_size = (3,3), activation ='relu', kernel_regularizer = 'l2'))
model.add(MaxPool2D(pool_size=(2,2)))

model.add(Conv2D(filters = 16, kernel_size = (7,7), activation ='relu', kernel_regularizer = 'l2'))
model.add(MaxPool2D(pool_size=(2,2)))

model.add(Conv2D(filters = 8, kernel_size = (5,5), activation ='relu', kernel_regularizer = 'l2'))
model.add(MaxPool2D(pool_size=(2,2)))

model.add(Flatten())
model.add(Dense(128, activation = "relu", kernel_regularizer = 'l2'))
model.add(Dense(64, activation = "relu", kernel_regularizer = 'l2'))
model.add(Dense(len(CLASS_NAMES), activation = "softmax"))

model.compile(loss = 'categorical_crossentropy', optimizer = Adam(0.0001),metrics=['accuracy'])

model.summary()
```

```
Model: "sequential"
```

```
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 220, 220, 64)      4864      
                                                                 
 max_pooling2d (MaxPooling2D  (None, 110, 110, 64)     0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 108, 108, 32)      18464     
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 54, 54, 32)       0         
 2D)                                                             
                                                                 
 conv2d_2 (Conv2D)           (None, 48, 48, 16)        25104     
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 24, 24, 16)       0         
 2D)                                                             
                                                                 
 conv2d_3 (Conv2D)           (None, 20, 20, 8)         3208      
                                                                 
 max_pooling2d_3 (MaxPooling  (None, 10, 10, 8)        0         
 2D)                                                             
                                                                 
 flatten (Flatten)           (None, 800)               0         
                                                                 
 dense (Dense)               (None, 128)               102528    
                                                                 
 dense_1 (Dense)             (None, 64)                8256      
                                                                 
 dense_2 (Dense)             (None, 3)                 195       
                                                                 
=================================================================
Total params: 162,619
Trainable params: 162,619
Non-trainable params: 0
_________________________________________________________________
```

Next we will create a network architecture for the model. We have used different types of layers according to their features namely Conv_2d (It is used to create a convolutional kernel that is convolved with the input layer to produce the output tensor), max_pooling2d (It is a downsampling technique which takes out the maximum value over the window defined by poolsize), flatten (It flattens the input and creates a 1D output), Dense (Dense layer produce the output as the dot product of input and kernel).

After defining the network architecture we found out the total parameters as 162,619.

```
# Splitting the data set into training and testing data sets
X_train_and_val, X_test, Y_train_and_val, Y_test = train_test_split(X_data, Y_data, test_size = 0.1)
# Splitting the training data set into training and validation data sets
X_train, X_val, Y_train, Y_val = train_test_split(X_train_and_val, Y_train_and_val, test_size = 0.2)
```

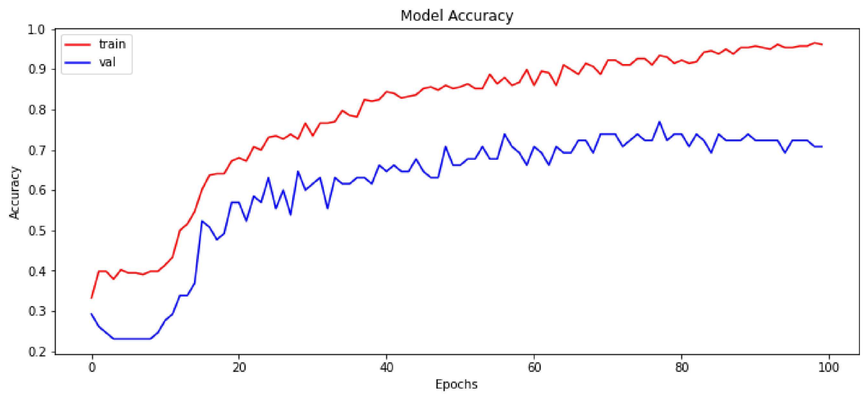
After defining the network architecture we will start with splitting the test and train data then dividing train data in train and validation data.

```
# Training the model
epochs = 100
batch_size = 128

history = model.fit(X_train, Y_train, batch_size = batch_size, epochs = epochs,
                    validation_data = (X_val, Y_val))
```

```
2/2 [==============================] - 45s 22s/step - loss: 2.9372 - accuracy: 0.9414 - val_loss: 3.2917 - val_accuracy: 0.7231
Epoch 85/100
2/2 [==============================] - 44s 22s/step - loss: 2.9098 - accuracy: 0.9453 - val_loss: 3.3221 - val_accuracy: 0.6923
Epoch 86/100
2/2 [==============================] - 47s 25s/step - loss: 2.9076 - accuracy: 0.9375 - val_loss: 3.2337 - val_accuracy: 0.7385
Epoch 87/100
2/2 [==============================] - 46s 27s/step - loss: 2.8866 - accuracy: 0.9492 - val_loss: 3.2602 - val_accuracy: 0.7231
Epoch 88/100
2/2 [==============================] - 44s 24s/step - loss: 2.8717 - accuracy: 0.9375 - val_loss: 3.2590 - val_accuracy: 0.7231
Epoch 89/100
2/2 [==============================] - 44s 24s/step - loss: 2.8587 - accuracy: 0.9531 - val_loss: 3.2160 - val_accuracy: 0.7231
Epoch 90/100
2/2 [==============================] - 44s 24s/step - loss: 2.8444 - accuracy: 0.9531 - val_loss: 3.2383 - val_accuracy: 0.7385
Epoch 91/100
2/2 [==============================] - 44s 23s/step - loss: 2.8354 - accuracy: 0.9570 - val_loss: 3.2208 - val_accuracy: 0.7231
Epoch 92/100
2/2 [==============================] - 44s 22s/step - loss: 2.8226 - accuracy: 0.9531 - val_loss: 3.2298 - val_accuracy: 0.7231
Epoch 93/100
2/2 [==============================] - 44s 23s/step - loss: 2.8140 - accuracy: 0.9492 - val_loss: 3.2056 - val_accuracy: 0.7231
Epoch 94/100
2/2 [==============================] - 44s 22s/step - loss: 2.8017 - accuracy: 0.9609 - val_loss: 3.1841 - val_accuracy: 0.7231
Epoch 95/100
2/2 [==============================] - 44s 22s/step - loss: 2.7890 - accuracy: 0.9531 - val_loss: 3.2314 - val_accuracy: 0.6923
Epoch 96/100
2/2 [==============================] - 47s 25s/step - loss: 2.7756 - accuracy: 0.9531 - val_loss: 3.1729 - val_accuracy: 0.7231
Epoch 97/100
2/2 [==============================] - 46s 26s/step - loss: 2.7655 - accuracy: 0.9570 - val_loss: 3.2061 - val_accuracy: 0.7231
Epoch 98/100
2/2 [==============================] - 44s 25s/step - loss: 2.7530 - accuracy: 0.9570 - val_loss: 3.1767 - val_accuracy: 0.7231
Epoch 99/100
2/2 [==============================] - 47s 27s/step - loss: 2.7501 - accuracy: 0.9648 - val_loss: 3.1662 - val_accuracy: 0.7077
Epoch 100/100
2/2 [==============================] - 44s 23s/step - loss: 2.7255 - accuracy: 0.9609 - val_loss: 3.2330 - val_accuracy: 0.7077
```

Now we will train our model on 100 epochs and a batch size of 128. You can try using more number of epochs to increase accuracy. During each epochs we can see how the model is performing by viewing the training and validation accuracy.

```
# Plot the training history
plt.figure(figsize=(12, 5))
plt.plot(history.history['accuracy'], color='r')
plt.plot(history.history['val_accuracy'], color='b')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['train', 'val'])

plt.show()
```



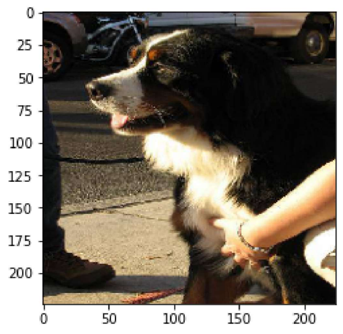
Here we analyse how the model is learning with each epoch in terms of accuracy.

```
Y_pred = model.predict(X_test)
score = model.evaluate(X_test, Y_test)
print('Accuracy over the test set: \n ', round((score[1]*100), 2), '%')
```

```
2/2 [==============================] - 2s 156ms/step
2/2 [==============================] - 2s 274ms/step - loss: 3.2658 - accuracy: 0.7222
Accuracy over the test set:
  72.22 %
```

We will use predict function to make predictions using this model also we are finding out the accuracy on the test set.

```
# Plotting image to compare
plt.imshow(X_test[1,:,:,:])
plt.show()

# Finding max value from predition list and comaparing original value vs predicted
print("Originally : ",labels['breed'][np.argmax(Y_test[1])])
print("Predicted : ",labels['breed'][np.argmax(Y_pred[1])])
```



```
Originally :  scottish_deerhound
Predicted :  scottish_deerhound
```

Here you can see image with its original and predicted label.

Conclusion:

We started with downloading the dataset creating the model and finding out the predictions using the model. We can optimize different hyper parameters in order to tune this model for a higher accuracy. This model can be used to predict different breeds of dogs which can be further used by different NGO's working on saving animals and for educational purposes also.

```
#Let's save the model for Deployment

model.save("dog_breed.h5")
```

✓ 0s completed at 01:40