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.

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

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

```
Mounted at /content/drive
```

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).

```
file_path = '/content/drive/MyDrive/40_datascience_project/Day2 Dog Breed Prediction/kaggle.json'
```

```
# 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.


# Step 1: Make sure the ~/.kaggle folder exists
!mkdir -p ~/.kaggle

# Step 2: Copy the kaggle.json from Google Drive to ~/.kaggle
!cp '/content/drive/MyDrive/40_datascience_project/Day2 Dog Breed Prediction/kaggle.json' ~/.kaggle/

# Step 3: Change the permissions to avoid warnings
!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
```

```
ref                                title                        size  lastUpdated                 downloadCount  voteCo
---------------------------------  ----------------------  ----------  --------------------------  -------------  ------
catherinehorng/dogbreedidfromcomp  dog-breed-id-from-comp   724495926  2020-06-26 03:09:05.433000           7901
```

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 ..
```

```
Dataset URL: https://www.kaggle.com/datasets/catherinehorng/dogbreedidfromcomp
License(s): unknown
Downloading dogbreedidfromcomp.zip to /content/dog_dataset
 98% 678M/691M [00:05<00:00, 170MB/s]
100% 691M/691M [00:05<00:00, 135MB/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/fe78fc42e32174c7178b572bdcf5a129.jpg
  inflating: dog_dataset/train/fe7ea4eb63ab5fddea120555790f9187.jpg
  inflating: dog_dataset/train/fe8d52ab96ff238ea7d234b508010ece.jpg
  inflating: dog_dataset/train/fe9e09be6594f626f0d711bfba10cfe0.jpg
  inflating: dog_dataset/train/fea60fdd28de5834520134d6dc77a9a2.jpg
  inflating: dog_dataset/train/feafd0730eae85e63a41bbc030755c59.jpg
  inflating: dog_dataset/train/feb16cf86c9dac6d476e3c372ba5c279.jpg
  inflating: dog_dataset/train/feb9d0ae525ca28aabff74b455e34c16.jpg
  inflating: dog_dataset/train/febcab8eb2da444bf83336cffec7eb92.jpg
  inflating: dog_dataset/train/fede60fb2acc02a2da0d0a05f760b7d5.jpg
  inflating: dog_dataset/train/fee1696ae6725863f84b0da2c05ad892.jpg
  inflating: dog_dataset/train/fee672d906b502642597ccbc6acff0bb.jpg
  inflating: dog_dataset/train/fee98c990f4d69c6a8467dd0f0668440.jpg
  inflating: dog_dataset/train/fef4a58219c8971820a85868a7b073f5.jpg
  inflating: dog_dataset/train/fef5d4cdaf50cf159102e803c7d6aa9c.jpg
  inflating: dog_dataset/train/fef9c3ab585ad3f778c549fda42c1856.jpg
  inflating: dog_dataset/train/fefb453e43ec5e840c323538261493bd.jpg
  inflating: dog_dataset/train/ff04baf19edbe449b39619d88da3633c.jpg
  inflating: dog_dataset/train/ff05f3976c17fef275cc0306965b3fe4.jpg
  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
from keras.preprocessing import 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

Next steps: Generate code with labels_all | View recommended plots | New interactive sheet

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()
```

breed	count
scottish_deerhound	126
maltese_dog	117
afghan_hound	116
entlebucher	115
bernese_mountain_dog	114

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

Next steps: Generate code with labels | View recommended plots | New interactive sheet

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

# 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 [00:02<00:00, 127.37it/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()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `in
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 220, 220, 64)	4,864
max_pooling2d (MaxPooling2D)	(None, 110, 110, 64)	0
conv2d_1 (Conv2D)	(None, 108, 108, 32)	18,464
max_pooling2d_1 (MaxPooling2D)	(None, 54, 54, 32)	0
conv2d_2 (Conv2D)	(None, 48, 48, 16)	25,104
max_pooling2d_2 (MaxPooling2D)	(None, 24, 24, 16)	0
conv2d_3 (Conv2D)	(None, 20, 20, 8)	3,208
max_pooling2d_3 (MaxPooling2D)	(None, 10, 10, 8)	0
flatten (Flatten)	(None, 800)	0
dense (Dense)	(None, 128)	102,528
dense_1 (Dense)	(None, 64)	8,256
dense_2 (Dense)	(None, 3)	195

Total params: 162,619 (635.23 KB)
Trainable params: 162,619 (635.23 KB)
Non-trainable params: 0 (0.00 B)

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)
```

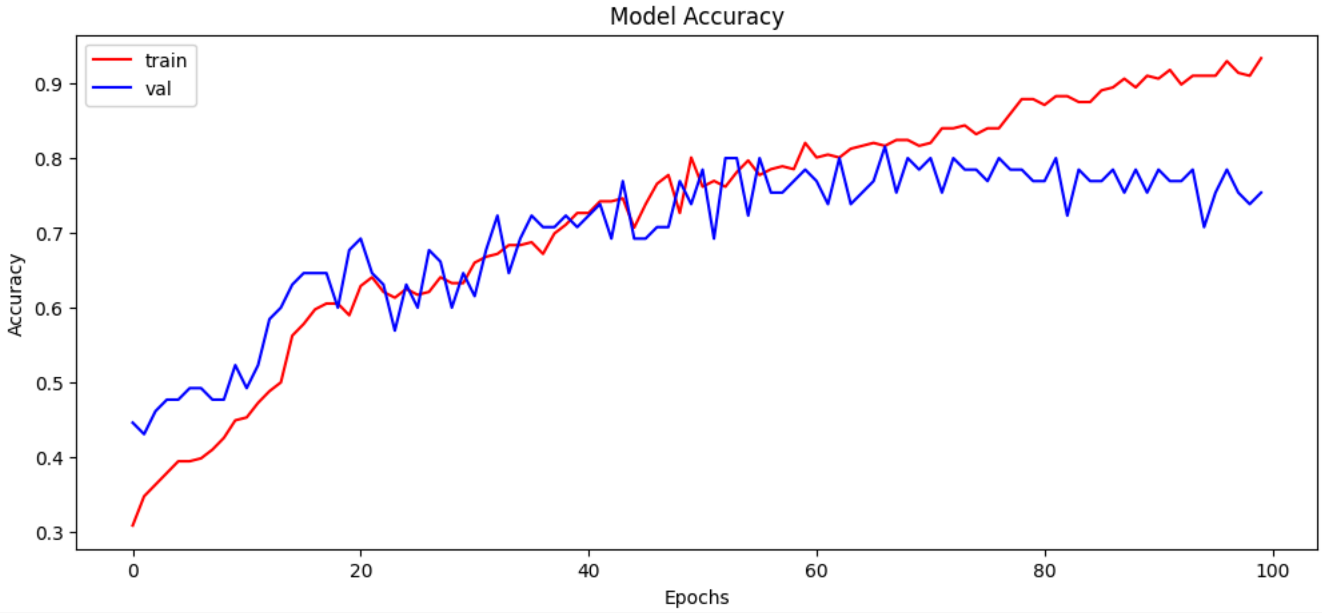
After defining the network architecture we will start with splitting the test and train data then dividing train data in train and validation data.

```
# 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))
```

```
Epoch 72/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 300ms/step - accuracy: 0.8411 - loss: 3.0884 - val_accuracy: 0.7538 - val_loss: 3.2539
Epoch 73/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 298ms/step - accuracy: 0.8307 - loss: 3.0828 - val_accuracy: 0.8000 - val_loss: 3.2309
Epoch 74/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 372ms/step - accuracy: 0.8411 - loss: 3.0666 - val_accuracy: 0.7846 - val_loss: 3.2384
Epoch 75/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 390ms/step - accuracy: 0.8333 - loss: 3.0585 - val_accuracy: 0.7846 - val_loss: 3.2109
Epoch 76/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 390ms/step - accuracy: 0.8464 - loss: 3.0153 - val_accuracy: 0.7692 - val_loss: 3.2119
Epoch 77/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 370ms/step - accuracy: 0.8542 - loss: 2.9902 - val_accuracy: 0.8000 - val_loss: 3.2072
Epoch 78/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 305ms/step - accuracy: 0.8542 - loss: 2.9890 - val_accuracy: 0.7846 - val_loss: 3.1822
Epoch 79/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 302ms/step - accuracy: 0.8750 - loss: 2.9906 - val_accuracy: 0.7846 - val_loss: 3.1862
Epoch 80/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 297ms/step - accuracy: 0.8906 - loss: 2.9438 - val_accuracy: 0.7692 - val_loss: 3.1704
Epoch 81/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 300ms/step - accuracy: 0.8724 - loss: 2.9445 - val_accuracy: 0.7692 - val_loss: 3.1715
Epoch 82/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 300ms/step - accuracy: 0.8854 - loss: 2.9211 - val_accuracy: 0.8000 - val_loss: 3.1425
Epoch 83/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 391ms/step - accuracy: 0.8906 - loss: 2.9080 - val_accuracy: 0.7231 - val_loss: 3.1758
Epoch 84/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 392ms/step - accuracy: 0.8776 - loss: 2.9150 - val_accuracy: 0.7846 - val_loss: 3.1228
Epoch 85/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 399ms/step - accuracy: 0.8542 - loss: 2.9012 - val_accuracy: 0.7692 - val_loss: 3.1538
Epoch 86/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 297ms/step - accuracy: 0.8984 - loss: 2.8616 - val_accuracy: 0.7692 - val_loss: 3.1288
Epoch 87/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 374ms/step - accuracy: 0.9036 - loss: 2.8464 - val_accuracy: 0.7846 - val_loss: 3.1001
Epoch 88/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 372ms/step - accuracy: 0.9167 - loss: 2.8240 - val_accuracy: 0.7538 - val_loss: 3.1579
Epoch 89/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 297ms/step - accuracy: 0.9010 - loss: 2.8266 - val_accuracy: 0.7846 - val_loss: 3.0980
Epoch 90/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 337ms/step - accuracy: 0.9010 - loss: 2.8121 - val_accuracy: 0.7538 - val_loss: 3.1144
Epoch 91/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 403ms/step - accuracy: 0.9062 - loss: 2.7955 - val_accuracy: 0.7846 - val_loss: 3.0728
Epoch 92/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 298ms/step - accuracy: 0.9375 - loss: 2.7679 - val_accuracy: 0.7692 - val_loss: 3.0881
Epoch 93/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 375ms/step - accuracy: 0.9115 - loss: 2.7855 - val_accuracy: 0.7692 - val_loss: 3.1273
Epoch 94/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 372ms/step - accuracy: 0.9167 - loss: 2.7691 - val_accuracy: 0.7846 - val_loss: 3.0521
Epoch 95/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 299ms/step - accuracy: 0.9010 - loss: 2.7731 - val_accuracy: 0.7077 - val_loss: 3.1722
Epoch 96/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 371ms/step - accuracy: 0.8984 - loss: 2.7679 - val_accuracy: 0.7538 - val_loss: 3.0678
Epoch 97/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 299ms/step - accuracy: 0.9245 - loss: 2.7438 - val_accuracy: 0.7846 - val_loss: 3.0385
Epoch 98/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 297ms/step - accuracy: 0.9219 - loss: 2.6944 - val_accuracy: 0.7538 - val_loss: 3.0723
Epoch 99/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 379ms/step - accuracy: 0.9010 - loss: 2.7010 - val_accuracy: 0.7385 - val_loss: 3.0512
Epoch 100/100
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 374ms/step - accuracy: 0.9323 - loss: 2.6732 - val_accuracy: 0.7538 - val_loss: 3.0252
```

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()
```



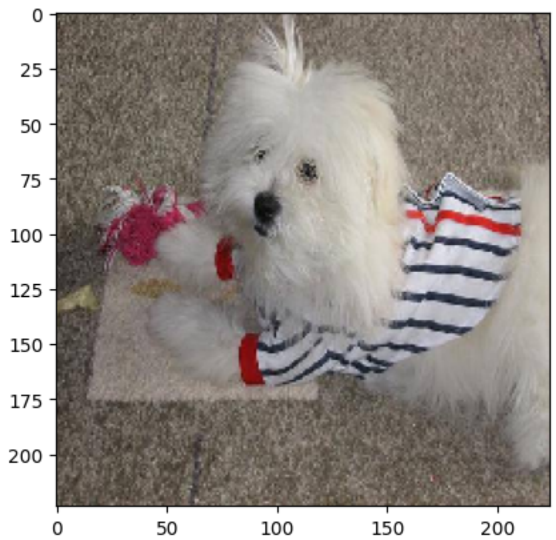
Here we analyse how the model is learning with each epoch in terms of accuracy.

```
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 692ms/step
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 574ms/step - accuracy: 0.6736 - loss: 3.2669
Accuracy over the test set: 
  66.67 %
```

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 :  maltese_dog
Predicted :  maltese_dog
```

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.

```
Start coding or generate with AI.
```