Important liprary imports

```
# Importing library
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
```

Loading the labels data into dataframe and viewing the data.

```
# Read the labels.csv file and check shape and records
labels_all = pd.read_csv('./dogbreedidfromcomp/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 number or each breed
breed_all = labels_all['breed']
breed_count = breed_all.value_counts()
breed_count.head()
```

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

```
# Selecting all breeds because i have high computation power
CLASS_NAME = ['scottish_deerhound', 'maltese_dog', 'afghan_hound',
'entlebucher', 'bernese_mountain_dog']
labels = labels_all[(labels_all['breed'].isin(CLASS_NAME))]
```

```
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	80	01ee3c7ff9bcaba9874183135877670e	entlebucher
4	88	021b5a49189665c0442c19b5b33e8cf1	entlebucher

```
# 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_NAME)

# Reading and converting image to numpy array and normalizing dataset
for i in tqdm(range(len(labels))):
    img = image.load_img('./dogbreedidfromcomp/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%|██████████| 588/588 [00:03<00:00, 169.31it/s]


Train Images shape:  (588, 224, 224, 3)  size: 88,510,464
One-hot encoded output shape:  (588, 5)  size: 2,940
```

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).

```
# 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_NAME), activation = "softmax"))

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

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
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 (MaxPooling2 (None, 54, 54, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 48, 48, 16)        25104
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 24, 24, 16)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 20, 20, 8)         3208
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 10, 10, 8)         0
_________________________________________________________________
flatten (Flatten)            (None, 800)               0
_________________________________________________________________
dense (Dense)                (None, 128)               102528
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
```

```
dense_2 (Dense)              (None, 5)                 325
=================================================================
Total params: 162,749
Trainable params: 162,749
Non-trainable params: 0
_________________________________________________________________
```

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

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

```
# 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)

# 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 1/100
4/4 [==============================] - 14s 3s/step - loss: 5.4002 -
accuracy: 0.1820 - val_loss: 5.3620 - val_accuracy: 0.2264
Epoch 2/100
4/4 [==============================] - 13s 3s/step - loss: 5.3482 -
accuracy: 0.2317 - val_loss: 5.3152 - val_accuracy: 0.1887
Epoch 3/100
4/4 [==============================] - 13s 3s/step - loss: 5.3004 -
accuracy: 0.2671 - val_loss: 5.2685 - val_accuracy: 0.1981
Epoch 4/100
4/4 [==============================] - 13s 3s/step - loss: 5.2535 -
accuracy: 0.2577 - val_loss: 5.2219 - val_accuracy: 0.2170
Epoch 5/100
4/4 [==============================] - 12s 3s/step - loss: 5.2060 -
accuracy: 0.2742 - val_loss: 5.1752 - val_accuracy: 0.2075
Epoch 6/100
4/4 [==============================] - 12s 3s/step - loss: 5.1590 -
accuracy: 0.3191 - val_loss: 5.1297 - val_accuracy: 0.2170
Epoch 7/100
4/4 [==============================] - 12s 3s/step - loss: 5.1119 -
```

```
accuracy: 0.2577 - val_loss: 5.0878 - val_accuracy: 0.2075
Epoch 8/100
4/4 [==============================] - 12s 3s/step - loss: 5.0658 -
accuracy: 0.2364 - val_loss: 5.0436 - val_accuracy: 0.2075
Epoch 9/100
4/4 [==============================] - 12s 3s/step - loss: 5.0194 -
accuracy: 0.2364 - val_loss: 4.9988 - val_accuracy: 0.2075
Epoch 10/100
4/4 [==============================] - 12s 3s/step - loss: 4.9739 -
accuracy: 0.2837 - val_loss: 4.9527 - val_accuracy: 0.2830
Epoch 11/100
4/4 [==============================] - 12s 3s/step - loss: 4.9263 -
accuracy: 0.3073 - val_loss: 4.9129 - val_accuracy: 0.2264
Epoch 12/100
4/4 [==============================] - 12s 3s/step - loss: 4.8787 -
accuracy: 0.3144 - val_loss: 4.8664 - val_accuracy: 0.2830
Epoch 13/100
4/4 [==============================] - 13s 3s/step - loss: 4.8306 -
accuracy: 0.3428 - val_loss: 4.8197 - val_accuracy: 0.2642
Epoch 14/100
4/4 [==============================] - 13s 3s/step - loss: 4.7806 -
accuracy: 0.3593 - val_loss: 4.7762 - val_accuracy: 0.2642
Epoch 15/100
4/4 [==============================] - 12s 3s/step - loss: 4.7283 -
accuracy: 0.3381 - val_loss: 4.7180 - val_accuracy: 0.3113
Epoch 16/100
4/4 [==============================] - 12s 3s/step - loss: 4.6755 -
accuracy: 0.3593 - val_loss: 4.6568 - val_accuracy: 0.3302
Epoch 17/100
4/4 [==============================] - 13s 3s/step - loss: 4.6091 -
accuracy: 0.3901 - val_loss: 4.6165 - val_accuracy: 0.3019
Epoch 18/100
4/4 [==============================] - 12s 3s/step - loss: 4.5533 -
accuracy: 0.4255 - val_loss: 4.5221 - val_accuracy: 0.3679
Epoch 19/100
4/4 [==============================] - 12s 3s/step - loss: 4.4802 -
accuracy: 0.4232 - val_loss: 4.4684 - val_accuracy: 0.3679
Epoch 20/100
4/4 [==============================] - 13s 3s/step - loss: 4.4286 -
accuracy: 0.4397 - val_loss: 4.3929 - val_accuracy: 0.3868
Epoch 21/100
4/4 [==============================] - 12s 3s/step - loss: 4.3663 -
accuracy: 0.4397 - val_loss: 4.3431 - val_accuracy: 0.4623
Epoch 22/100
4/4 [==============================] - 12s 3s/step - loss: 4.2817 -
accuracy: 0.4917 - val_loss: 4.2840 - val_accuracy: 0.4717
Epoch 23/100
4/4 [==============================] - 12s 3s/step - loss: 4.2441 -
accuracy: 0.4563 - val_loss: 4.2103 - val_accuracy: 0.4528
```

```
Epoch 24/100
4/4 [==============================] - 12s 3s/step - loss: 4.1939 -
accuracy: 0.4988 - val_loss: 4.1895 - val_accuracy: 0.4528
Epoch 25/100
4/4 [==============================] - 12s 3s/step - loss: 4.1270 -
accuracy: 0.5035 - val_loss: 4.1390 - val_accuracy: 0.4528
Epoch 26/100
4/4 [==============================] - 12s 3s/step - loss: 4.0735 -
accuracy: 0.5154 - val_loss: 4.0954 - val_accuracy: 0.4623
Epoch 27/100
4/4 [==============================] - 12s 3s/step - loss: 4.0227 -
accuracy: 0.5201 - val_loss: 4.0544 - val_accuracy: 0.4717
Epoch 28/100
4/4 [==============================] - 13s 3s/step - loss: 3.9889 -
accuracy: 0.5225 - val_loss: 4.0310 - val_accuracy: 0.4623
Epoch 29/100
4/4 [==============================] - 13s 3s/step - loss: 3.9455 -
accuracy: 0.5390 - val_loss: 3.9880 - val_accuracy: 0.4528
Epoch 30/100
4/4 [==============================] - 13s 3s/step - loss: 3.8986 -
accuracy: 0.5556 - val_loss: 3.9746 - val_accuracy: 0.4623
Epoch 31/100
4/4 [==============================] - 12s 3s/step - loss: 3.8653 -
accuracy: 0.5414 - val_loss: 3.9091 - val_accuracy: 0.4528
Epoch 32/100
4/4 [==============================] - 12s 3s/step - loss: 3.8045 -
accuracy: 0.5674 - val_loss: 3.8863 - val_accuracy: 0.4623
Epoch 33/100
4/4 [==============================] - 12s 3s/step - loss: 3.7706 -
accuracy: 0.5697 - val_loss: 3.8699 - val_accuracy: 0.4623
Epoch 34/100
4/4 [==============================] - 12s 3s/step - loss: 3.7393 -
accuracy: 0.5863 - val_loss: 3.8311 - val_accuracy: 0.4623
Epoch 35/100
4/4 [==============================] - 12s 3s/step - loss: 3.6918 -
accuracy: 0.5981 - val_loss: 3.8209 - val_accuracy: 0.4717
Epoch 36/100
4/4 [==============================] - 12s 3s/step - loss: 3.6551 -
accuracy: 0.6076 - val_loss: 3.8011 - val_accuracy: 0.4811
Epoch 37/100
4/4 [==============================] - 12s 3s/step - loss: 3.6250 -
accuracy: 0.6312 - val_loss: 3.7678 - val_accuracy: 0.4906
Epoch 38/100
4/4 [==============================] - 12s 3s/step - loss: 3.5799 -
accuracy: 0.6430 - val_loss: 3.7603 - val_accuracy: 0.4906
Epoch 39/100
4/4 [==============================] - 12s 3s/step - loss: 3.5719 -
accuracy: 0.6194 - val_loss: 3.7886 - val_accuracy: 0.4623
Epoch 40/100
```

```
4/4 [==============================] - 12s 3s/step - loss: 3.5856 -
accuracy: 0.6501 - val_loss: 3.7515 - val_accuracy: 0.5094
Epoch 41/100
4/4 [==============================] - 12s 3s/step - loss: 3.5127 -
accuracy: 0.6572 - val_loss: 3.7295 - val_accuracy: 0.4717
Epoch 42/100
4/4 [==============================] - 12s 3s/step - loss: 3.4867 -
accuracy: 0.6548 - val_loss: 3.6771 - val_accuracy: 0.5000
Epoch 43/100
4/4 [==============================] - 12s 3s/step - loss: 3.4377 -
accuracy: 0.6785 - val_loss: 3.6510 - val_accuracy: 0.5283
Epoch 44/100
4/4 [==============================] - 12s 3s/step - loss: 3.4157 -
accuracy: 0.6478 - val_loss: 3.6823 - val_accuracy: 0.5189
Epoch 45/100
4/4 [==============================] - 12s 3s/step - loss: 3.3979 -
accuracy: 0.6974 - val_loss: 3.6133 - val_accuracy: 0.5660
Epoch 46/100
4/4 [==============================] - 12s 3s/step - loss: 3.3516 -
accuracy: 0.6761 - val_loss: 3.6109 - val_accuracy: 0.4906
Epoch 47/100
4/4 [==============================] - 12s 3s/step - loss: 3.3524 -
accuracy: 0.6690 - val_loss: 3.6389 - val_accuracy: 0.5189
Epoch 48/100
4/4 [==============================] - 13s 3s/step - loss: 3.3508 -
accuracy: 0.6856 - val_loss: 3.7187 - val_accuracy: 0.4623
Epoch 49/100
4/4 [==============================] - 12s 3s/step - loss: 3.3118 -
accuracy: 0.6998 - val_loss: 3.6145 - val_accuracy: 0.5283
Epoch 50/100
4/4 [==============================] - 12s 3s/step - loss: 3.2780 -
accuracy: 0.6950 - val_loss: 3.5596 - val_accuracy: 0.5283
Epoch 51/100
4/4 [==============================] - 12s 3s/step - loss: 3.2462 -
accuracy: 0.6832 - val_loss: 3.5568 - val_accuracy: 0.5094
Epoch 52/100
4/4 [==============================] - 12s 3s/step - loss: 3.2210 -
accuracy: 0.7376 - val_loss: 3.5354 - val_accuracy: 0.5283
Epoch 53/100
4/4 [==============================] - 12s 3s/step - loss: 3.2411 -
accuracy: 0.6785 - val_loss: 3.6600 - val_accuracy: 0.4717
Epoch 54/100
4/4 [==============================] - 12s 3s/step - loss: 3.2740 -
accuracy: 0.6879 - val_loss: 3.5986 - val_accuracy: 0.5283
Epoch 55/100
4/4 [==============================] - 12s 3s/step - loss: 3.2298 -
accuracy: 0.6856 - val_loss: 3.5963 - val_accuracy: 0.5000
Epoch 56/100
4/4 [==============================] - 13s 3s/step - loss: 3.1483 -
```

```
accuracy: 0.7494 - val_loss: 3.5096 - val_accuracy: 0.5189
Epoch 57/100
4/4 [==============================] - 12s 3s/step - loss: 3.1530 -
accuracy: 0.7210 - val_loss: 3.5104 - val_accuracy: 0.5283
Epoch 58/100
4/4 [==============================] - 12s 3s/step - loss: 3.0804 -
accuracy: 0.7565 - val_loss: 3.4639 - val_accuracy: 0.5566
Epoch 59/100
4/4 [==============================] - 13s 3s/step - loss: 3.0406 -
accuracy: 0.7707 - val_loss: 3.4634 - val_accuracy: 0.5283
Epoch 60/100
4/4 [==============================] - 12s 3s/step - loss: 3.0289 -
accuracy: 0.7872 - val_loss: 3.4432 - val_accuracy: 0.5189
Epoch 61/100
4/4 [==============================] - 12s 3s/step - loss: 3.0041 -
accuracy: 0.7825 - val_loss: 3.4394 - val_accuracy: 0.5283
Epoch 62/100
4/4 [==============================] - 12s 3s/step - loss: 2.9789 -
accuracy: 0.7849 - val_loss: 3.4250 - val_accuracy: 0.5377
Epoch 63/100
4/4 [==============================] - 12s 3s/step - loss: 2.9453 -
accuracy: 0.8061 - val_loss: 3.4228 - val_accuracy: 0.5755
Epoch 64/100
4/4 [==============================] - 12s 3s/step - loss: 2.9287 -
accuracy: 0.8038 - val_loss: 3.4297 - val_accuracy: 0.5377
Epoch 65/100
4/4 [==============================] - 12s 3s/step - loss: 2.9285 -
accuracy: 0.7896 - val_loss: 3.4039 - val_accuracy: 0.5377
Epoch 66/100
4/4 [==============================] - 12s 3s/step - loss: 2.9073 -
accuracy: 0.7825 - val_loss: 3.4794 - val_accuracy: 0.5566
Epoch 67/100
4/4 [==============================] - 12s 3s/step - loss: 2.8829 -
accuracy: 0.8061 - val_loss: 3.4491 - val_accuracy: 0.5566
Epoch 68/100
4/4 [==============================] - 12s 3s/step - loss: 2.8765 -
accuracy: 0.7896 - val_loss: 3.5654 - val_accuracy: 0.4906
Epoch 69/100
4/4 [==============================] - 12s 3s/step - loss: 2.8764 -
accuracy: 0.8085 - val_loss: 3.4269 - val_accuracy: 0.5189
Epoch 70/100
4/4 [==============================] - 12s 3s/step - loss: 2.8112 -
accuracy: 0.8298 - val_loss: 3.4867 - val_accuracy: 0.5094
Epoch 71/100
4/4 [==============================] - 12s 3s/step - loss: 2.8119 -
accuracy: 0.8416 - val_loss: 3.4141 - val_accuracy: 0.5283
Epoch 72/100
4/4 [==============================] - 12s 3s/step - loss: 2.7762 -
accuracy: 0.8227 - val_loss: 3.4941 - val_accuracy: 0.5189
```

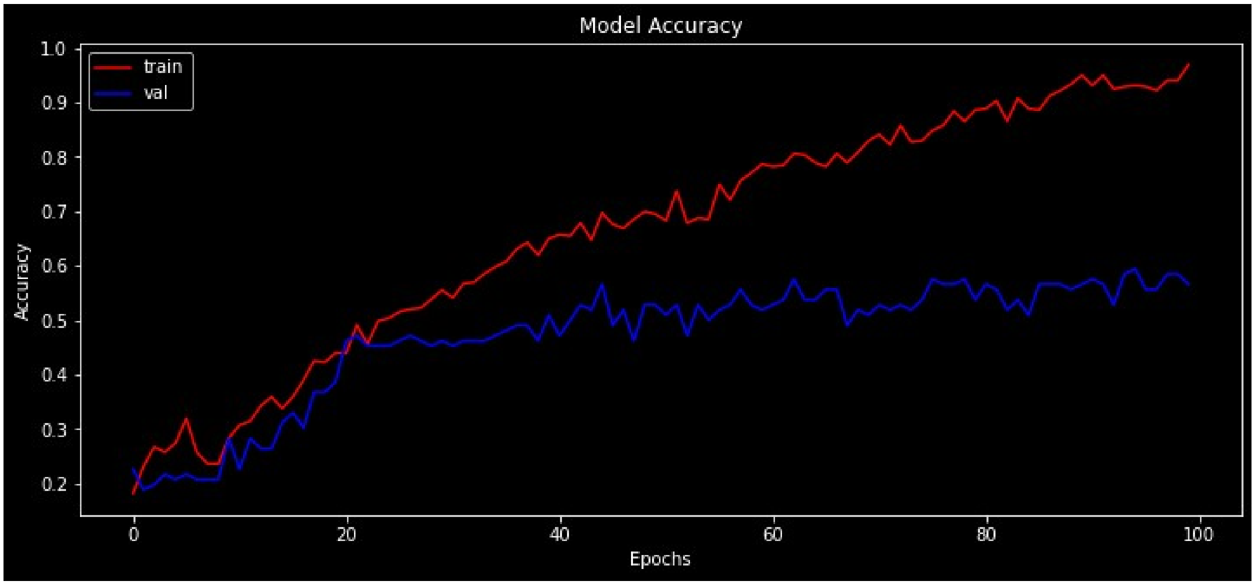
```
Epoch 73/100
4/4 [==============================] - 12s 3s/step - loss: 2.7460 -
accuracy: 0.8582 - val_loss: 3.4377 - val_accuracy: 0.5283
Epoch 74/100
4/4 [==============================] - 12s 3s/step - loss: 2.7577 -
accuracy: 0.8274 - val_loss: 3.5169 - val_accuracy: 0.5189
Epoch 75/100
4/4 [==============================] - 12s 3s/step - loss: 2.7408 -
accuracy: 0.8298 - val_loss: 3.4018 - val_accuracy: 0.5377
Epoch 76/100
4/4 [==============================] - 12s 3s/step - loss: 2.7228 -
accuracy: 0.8487 - val_loss: 3.3748 - val_accuracy: 0.5755
Epoch 77/100
4/4 [==============================] - 12s 3s/step - loss: 2.6835 -
accuracy: 0.8582 - val_loss: 3.3695 - val_accuracy: 0.5660
Epoch 78/100
4/4 [==============================] - 12s 3s/step - loss: 2.6567 -
accuracy: 0.8842 - val_loss: 3.3700 - val_accuracy: 0.5660
Epoch 79/100
4/4 [==============================] - 12s 3s/step - loss: 2.6286 -
accuracy: 0.8652 - val_loss: 3.3738 - val_accuracy: 0.5755
Epoch 80/100
4/4 [==============================] - 12s 3s/step - loss: 2.6028 -
accuracy: 0.8865 - val_loss: 3.3576 - val_accuracy: 0.5377
Epoch 81/100
4/4 [==============================] - 12s 3s/step - loss: 2.5874 -
accuracy: 0.8889 - val_loss: 3.3809 - val_accuracy: 0.5660
Epoch 82/100
4/4 [==============================] - 12s 3s/step - loss: 2.5681 -
accuracy: 0.9031 - val_loss: 3.3767 - val_accuracy: 0.5566
Epoch 83/100
4/4 [==============================] - 12s 3s/step - loss: 2.5879 -
accuracy: 0.8652 - val_loss: 3.4766 - val_accuracy: 0.5189
Epoch 84/100
4/4 [==============================] - 12s 3s/step - loss: 2.5548 -
accuracy: 0.9078 - val_loss: 3.3906 - val_accuracy: 0.5377
Epoch 85/100
4/4 [==============================] - 12s 3s/step - loss: 2.5424 -
accuracy: 0.8889 - val_loss: 3.4985 - val_accuracy: 0.5094
Epoch 86/100
4/4 [==============================] - 12s 3s/step - loss: 2.5550 -
accuracy: 0.8865 - val_loss: 3.3639 - val_accuracy: 0.5660
Epoch 87/100
4/4 [==============================] - 12s 3s/step - loss: 2.5120 -
accuracy: 0.9125 - val_loss: 3.3809 - val_accuracy: 0.5660
Epoch 88/100
4/4 [==============================] - 12s 3s/step - loss: 2.4737 -
accuracy: 0.9220 - val_loss: 3.3978 - val_accuracy: 0.5660
Epoch 89/100
```

```
4/4 [==============================] - 12s 3s/step - loss: 2.4522 -
accuracy: 0.9338 - val_loss: 3.3430 - val_accuracy: 0.5566
Epoch 90/100
4/4 [==============================] - 12s 3s/step - loss: 2.4259 -
accuracy: 0.9504 - val_loss: 3.3874 - val_accuracy: 0.5660
Epoch 91/100
4/4 [==============================] - 12s 3s/step - loss: 2.4192 -
accuracy: 0.9314 - val_loss: 3.4103 - val_accuracy: 0.5755
Epoch 92/100
4/4 [==============================] - 12s 3s/step - loss: 2.3986 -
accuracy: 0.9504 - val_loss: 3.4067 - val_accuracy: 0.5660
Epoch 93/100
4/4 [==============================] - 12s 3s/step - loss: 2.4084 -
accuracy: 0.9243 - val_loss: 3.4785 - val_accuracy: 0.5283
Epoch 94/100
4/4 [==============================] - 12s 3s/step - loss: 2.3986 -
accuracy: 0.9291 - val_loss: 3.4121 - val_accuracy: 0.5849
Epoch 95/100
4/4 [==============================] - 12s 3s/step - loss: 2.3758 -
accuracy: 0.9314 - val_loss: 3.3729 - val_accuracy: 0.5943
Epoch 96/100
4/4 [==============================] - 13s 3s/step - loss: 2.3691 -
accuracy: 0.9291 - val_loss: 3.5682 - val_accuracy: 0.5566
Epoch 97/100
4/4 [==============================] - 12s 3s/step - loss: 2.3980 -
accuracy: 0.9220 - val_loss: 3.3890 - val_accuracy: 0.5566
Epoch 98/100
4/4 [==============================] - 13s 3s/step - loss: 2.3404 -
accuracy: 0.9409 - val_loss: 3.3838 - val_accuracy: 0.5849
Epoch 99/100
4/4 [==============================] - 12s 3s/step - loss: 2.3159 -
accuracy: 0.9409 - val_loss: 3.4029 - val_accuracy: 0.5849
Epoch 100/100
4/4 [==============================] - 12s 3s/step - loss: 2.2830 -
accuracy: 0.9693 - val_loss: 3.4040 - val_accuracy: 0.5660
```

Here we analyse how the model is learning with each epoch in terms of 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()
```



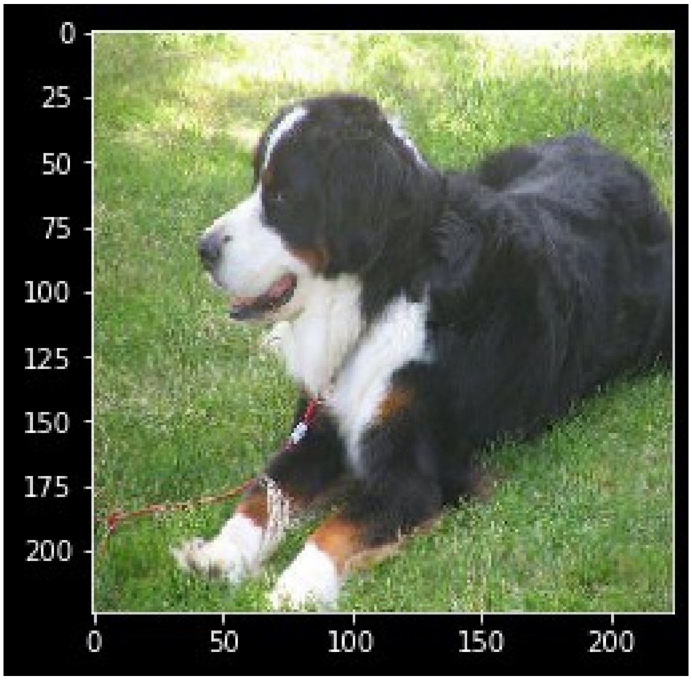
We will use predict function to make predictions using this model also we are finding out the accuracy on the test set.

```
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 [==============================] - 0s 102ms/step - loss: 3.8915 -
accuracy: 0.5424
Accuracy over the test set:
  54.24 %

# 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 :  entlebucher
Predicted :  entlebucher
```

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.