In [5]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow import keras
import pickle
from dataloader import tfdata_generator
from keras.models import load_model
import os
```

In [4]:
```
labels = pd.read_csv('labels.csv')
labels.head()
```

Out[4]:

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

In [6]:
```
files_path = []
for root, dirs, files in os.walk("train", topdown=False):
   for name in sorted(files):
      files_path.append(os.path.join(root, name))
```

In [7]:
```
files = pd.DataFrame(list(zip(files_path,labels.breed.to_list())),columns=['path','label'])
files.head()
```

Out[7]:

	path	label
0	train/000bec180eb18c7604dcecc8fe0dba07.jpg	boston_bull
1	train/001513dfcb2ffafc82cccf4d8bbaba97.jpg	dingo
2	train/001cdf01b096e06d78e9e5112d419397.jpg	pekinese
3	train/00214f311d5d2247d5dfe4fe24b2303d.jpg	bluetick
4	train/0021f9ceb3235effd7fcde7f7538ed62.jpg	golden_retriever

In [8]:
```
#Loading the encoder which is created while training.
with open('/content/drive/MyDrive/dog_breed/encoder','rb') as file:
    encoder = pickle.load(file)
```

In [9]:
```
image_labels = encoder.transform(files['label'].to_list())
```

In [10]:
```
image_paths  = files['path'].tolist()
```

In [11]:
```
image_data_generator = tfdata_generator(image_paths, image_labels, is_training=False)
```

In [12]:
```
#Loading the best model obtained after experimentation
Stacked_model_1 = load_model('/content/drive/MyDrive/dog_breed/Inception_resnet_stacked_weights.h5')
```

In [13]:
```
def modelling(data_generator,model=Stacked_model_1):
    """
    This function will return the Loss and Accuracy by evaluating the data using the best model.
    This function takes train data as tensor and trained model.
    """
    results         = model.evaluate(data_generator,steps=100,verbose=0)
    print('Loss: ',results[0])
    print('Accuracy: ',results[1])
    return None
modelling(image_data_generator)
```

```
Loss:  0.1003708466887474
Accuracy:  0.9706249833106995
```

In [14]:
```
test_files = []
for root, dirs, files in os.walk("test", topdown=False):
   for name in sorted(files):
      test_files.append(os.path.join(root, name))
```

In [15]:
```
test_image_path = test_files[0]
```

In [16]:
```
def estimate(test_image_path,model=Stacked_model_1):
    """
    This function will return the predictions for each dog breed label
    by taking the image and best model as the input.
    """
    image = tf.io.read_file(test_image_path)
    image = tf.image.decode_jpeg(image, channels=3)
    image = image/255
    image = tf.image.convert_image_dtype(image, tf.float32)
    image = tf.image.resize(image, [400, 400])
    predictions = model.predict(image[np.newaxis,:,:,:],verbose=0)
    return predictions
estimate(test_image_path)
```

Out[16]:
```
array([[2.13981949e-07, 1.09429372e-07, 8.95456296e-08, 1.00257083e-07,
        3.71944026e-07, 1.68061689e-08, 2.82568102e-08, 3.74299098e-08,
        1.00755123e-07, 4.33248175e-08, 2.38967264e-08, 5.71410013e-08,
        8.49853876e-08, 2.27446276e-06, 7.34796288e-07, 5.02358830e-07,
        2.92693187e-07, 2.56265924e-08, 7.31431626e-08, 6.30322745e-07,
        1.25154926e-08, 2.10035353e-07, 1.56985288e-06, 2.64991769e-07,
        4.32793925e-08, 2.14196052e-06, 3.56475567e-07, 4.01320079e-08,
        2.72786508e-07, 2.94776669e-06, 3.16347666e-07, 6.17105167e-09,
        3.75244980e-07, 3.38983938e-07, 3.27307902e-07, 1.49979962e-06,
        1.39323265e-06, 4.38105019e-09, 6.78227536e-07, 4.28247882e-09,
        2.79466462e-07, 3.17898703e-08, 6.08308071e-08, 9.54548707e-09,
        1.33346916e-07, 1.96643285e-07, 1.04513475e-07, 2.16150070e-07,
        6.23792857e-08, 1.04177877e-06, 1.56781439e-07, 4.94047697e-07,
        8.49129620e-08, 1.32094158e-08, 3.57634491e-08, 1.81291853e-08,
        4.89738841e-06, 7.56207328e-08, 2.34532589e-08, 2.63651838e-08,
        2.24527099e-08, 9.98915315e-01, 1.66896381e-08, 8.32304110e-08,
        7.12395121e-08, 2.27548540e-07, 2.51263884e-07, 1.44957923e-07,
        1.35459985e-07, 3.85751449e-08, 3.99500323e-07, 4.42862330e-07,
        1.67443133e-07, 2.71461568e-06, 6.41320810e-07, 2.04694985e-07,
        2.18815629e-07, 5.20136609e-07, 4.40689867e-08, 3.47516966e-08,
        1.44090009e-07, 4.27515161e-08, 1.22546425e-07, 3.22490621e-08,
        2.13560452e-06, 1.00575329e-03, 2.87706456e-07, 1.56010174e-07,
        7.37635617e-07, 1.68240746e-07, 1.62158176e-06, 6.43870450e-08,
        2.82288397e-07, 3.23727363e-07, 1.41815121e-07, 1.99424193e-07,
        8.77574067e-08, 3.77830283e-08, 4.20133404e-07, 1.58100408e-06,
        3.00330012e-05, 5.75637955e-08, 1.88120026e-08, 1.90882247e-07,
        1.45807547e-07, 2.89138427e-07, 1.82370030e-07, 3.24190310e-08,
        1.76751499e-07, 3.89571960e-06, 1.06400755e-07, 4.01414411e-07,
        1.62019358e-08, 9.64391091e-08, 1.01823218e-06, 7.00956022e-08,
        3.60562353e-07, 1.14011813e-08, 9.01388830e-08, 2.44355931e-07]],
      dtype=float32)
```