Dog Breed Classification

Siddamshetty Bhavesh Kumar and Ch.v.Chaitanya Reddy
bhavesh.20bci7220@vitap.ac.in, chaitanya.20bci7204@vitap.ac.in

***Abstract* - The objective of this project is to predict the breed of the dog from the given input images. For that this project uses computer vision and deep learning techniques to predict dog breeds from the images. Deep learning follows some steps to complete the task. First, these techniques identify the facial key features for each image(dog) by using a convolutional neural network. We take those attributes / features to a fully connected neural network layer and get the results. For our project first we went with one pretrained model, and then tried concatenating more of pre trained models i.e., Ensembling them and getting the results.Our highest accuracy was by stacking 3 pre trained models 94%.**

1. Introduction

1.1 Motivation

There are many breeds of dogs in this world. It's hard to remember and identify by seeing with our naked eye. Different people buy their dogs based on the breed of the dog. Some people may buy without knowing the breed of the dog. So if we want to buy a dog, we should know about the breed of it. Some information should be gathered like gathering the information about the dog breed that our neighbor has or anyone from the colony. Through this project we don't need to gather any sort of information because it finds the dog breed by taking the image of the dog as an input. This is done by a concept called Image Processing. This process is used to extract important features from the image, analyze them and train the model to predict the required output.

1.2 Convolution Neural Networks

Convolutional neural networks are used for image processing to obtain the best accuracy. Convolutional Neural networks have two parts, feature extraction(Convolution) and classification.
The first layer in a CNN network is the convolutional layer which is the core building block and does most of the computational heavy lifting. Data or image is convolved using filters or kernels. Filters are small units that we apply across the data through a sliding window. The depth of the image is the same as the input, for a color image whose RGB value is 4, a filter of depth 4 would also be applied to it. This process involves taking the element-wise product of filters in the image and then summing those specific values for every sliding action. The output of a convolution that has a 3d filter with color would be a 2d matrix.
Second is the activation layer which applies the ReLu (Rectified Linear Unit), in this step we apply the rectifier function to increase non-linearity in the CNN. Images are made of different objects that are not linear to each other.
Third, is the pooling layer, which involves downsampling of features. It is applied through every layer in the 3d volume. Typically there are hyperparameters within this layer:

1.3 Feature Extraction

Feature extraction is a part of the dimensionality reduction process, in which an initial set of the raw data is divided and reduced to more manageable groups. So when you want to process it will be easier. The most important characteristic of these large data sets is that they have a large number of variables. These variables require a lot of computing resources to process. So Feature extraction helps to get the best feature from those big data sets by selecting and combining variables into features, thus, effectively reducing the amount of data. These features are easy to process, but still able to describe the actual data set with accuracy and originality.

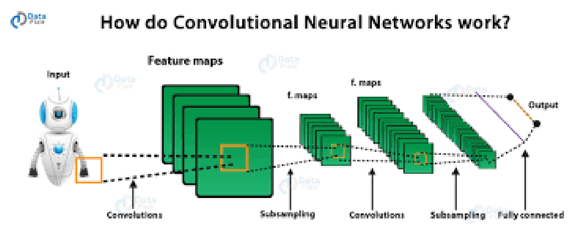


Fig1: How do Convolutional Neural Networks Work?

1.4 Transfer Learning

The reuse of a previously learned model on a new problem is known as transfer learning. It's particularly popular in deep learning right now since it can train deep neural networks with a small amount of data. This is particularly valuable in the field of data science, as most real-world situations do not require millions of labeled data points to train complicated models.

1.5 Pre Trained Model

A pre-trained model has been previously trained on a dataset and contains the weights and biases that represent the features of whichever dataset it was trained on. Learned features are often transferable to different data. For example, if you want to build a self learning car. You can spend years to build a decent image recognition algorithm from scratch or you can take the inception model (a pre-trained model) from Google which was built on ImageNet data to identify images in those pictures.A pre-trained model may not be 100% accurate in your application, but it saves huge efforts required to reinvent the wheel.

2.Literature Survey

There was research done on depth wise separable convolutions in neural computer vision architectures and their effectiveness when used in place of Inception modules. An architecture based on this idea was proposed called Xception which has the same parameter count as Inception V3 but is easy to use as regular convolution layers.
Another method proposed, suggested several design principles to scale up convolutional networks which contribute to high performance vision networks having relatively modest computation cost compared to simpler, more monolithic architectures. The study was done in the frame of reference of the Inception architecture and comparison was drawn between the error rates with the others proving the discussed method was efficient.
In another study, the architecture Inception-ResNet-v2 has been presented and discussed by the authors. There was also discussion regarding the variation of training speed with the introduction of residual connections for the Inception architecture.
In another work, the Transfer Learning approach was discussed with Deep CNN, as we faced the problem of collecting enough training data to rebuild models.
Their result reveals that transfer learning is an effective way to solve the data hungry problem.

One of the works used transfer learning with various settings on the dataset. For their dataset the transfer learning gave them a boosted performance of 89.92% accuracy.

In another work, research on ensemble learning was done. They saw how to combine the advantages of various models and construct a stronger classifier and it showed them great results.

Most of the previous works used hand-crafted features that would find it difficult to discriminate between a large number of breeds. These selected features have been limited to certain types and may not contain sufficient information to increase the classification between breeds. Unlike conventional techniques, deep learning can create different features during training from the original images and achieve significant results that will be explored in this work.

S.No	Paper Name	Models Used	Accuracy Obtained
1	Dog breed identification using convolution neural networks on android	Inception-V 3 Inception ResNet v2 and Xception	Highest obtained - 94%

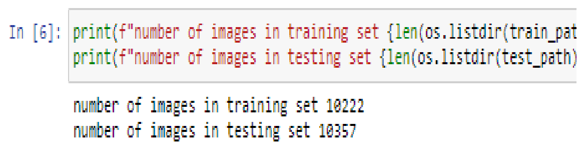
2	Knowing Your Dog Breed: Identifying a Dog Breed with Deep Learning	MobileNet V2, InceptionV3 , and NASNet	89.92%
3	Dog Breed Identification with a Neural Network over Learned Representations from The Xception CNN Architecture	Xception	78.58%
4	An Intelligent Dog breed recognition system using deep learning	Basic CNN MobileNet V2 Fine Tuning	90%

Table 1: Reference papers and their models and their accuracy.

3. My work

3.1 Dataset

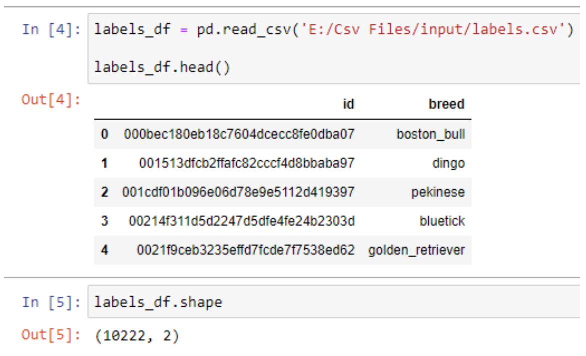
We have used a dataset consisting of 10222 images of dogs in the training dataset and 10357 images of dogs in the testing dataset. The model was only exposed to the training dataset with 10222 images. The testing dataset was used for validation and the accuracy of the model. In the training and testing dataset we have a total of 120 breeds of dogs.



```
In [6]: print(f"number of images in training set {len(os.listdir(train_pat
        print(f"number of images in testing set {len(os.listdir(test_path)

        number of images in training set 10222
        number of images in testing set 10357
```

Fig 2: Number of images in training and testing dataset

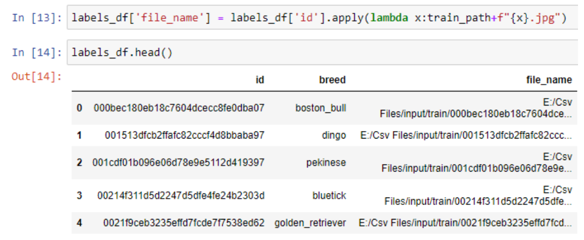
For labels of the dogs corresponding to the training set we have a labels data-frame which contains all the image id's and their respective breeds.



```
In [4]: labels_df = pd.read_csv('E:/Csv Files/input/labels.csv')

        labels_df.head()

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

In [5]: labels_df.shape

Out[5]: (10222, 2)
```

Fig 3: Data frame containing the id's and corresponding dog breeds

Then later a data frame was created where 3 columns represented the id of the image, with breed name and the file name of the image in the training folder.



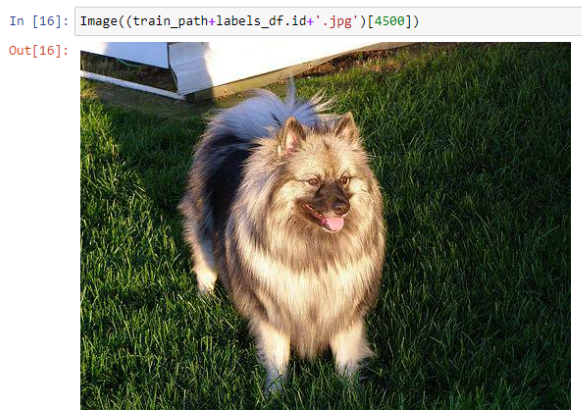
```
In [13]: labels_df['file_name'] = labels_df['id'].apply(lambda x:train_path+f"{x}.jpg")

In [14]: labels_df.head()

Out[14]:
                                     id             breed                                     file_name
        0  000bec180eb18c7604dcecc8fe0dba07       boston_bull                                        E:/Csv
                                                               Files/input/train/000bec180eb18c7604dce...
        1  001513dfcb2ffafc82cccf4d8bbaba97             dingo E:/Csv Files/input/train/001513dfcb2ffafc82ccc...
        2  001cdf01b096e06d78e9e5112d419397          pekinese                                        E:/Csv
                                                               Files/input/train/001cdf01b096e06d78e9e...
        3  00214f311d5d2247d5dfe4fe24b2303d          bluetick                                        E:/Csv
                                                               Files/input/train/00214f311d5d2247d5dfe...
        4  0021f9ceb3235effd7fcde7f7538ed62  golden_retriever E:/Csv Files/input/train/0021f9ceb3235effd7fcd...
```

Fig 4: Updated the previous data frame (Fig 3) with file names.

3.2 Visualizing the dataset

First we plot check some random dog images in the training set so that we confirm all of them.

```
In [14]: from IPython.display import display,Image
         Image((train_path+labels_df.id+'.jpg')[500])

Out[14]:
```



```
In [16]: Image((train_path+labels_df.id+'.jpg')[4500])
Out[16]:
```

Fig 5: Dog images from the training set.

Then we plot a graph to count the number of dogs of each breed available in the dataset.

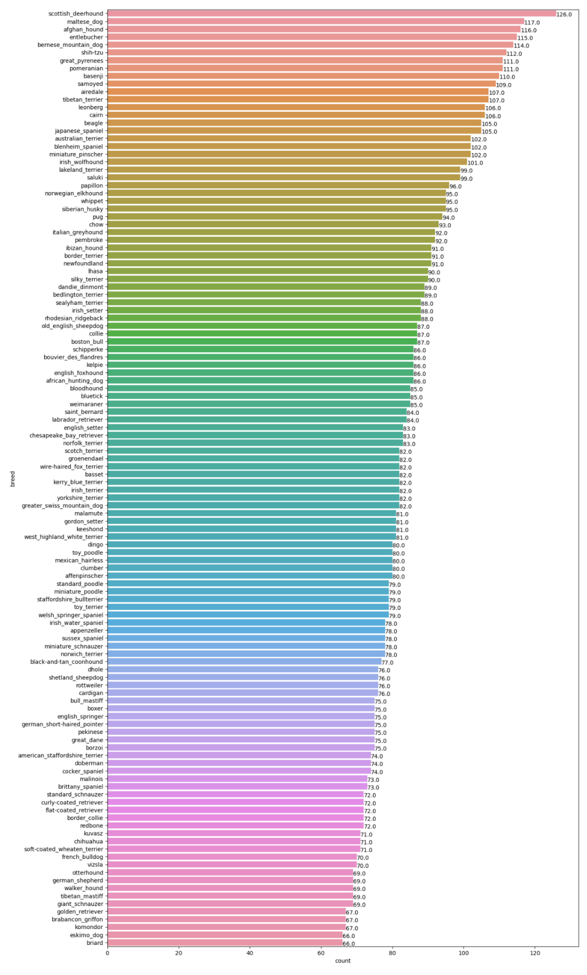
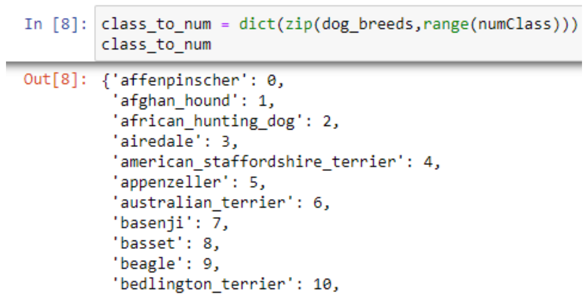


Fig 6: Graph that shows the number of dogs per breed From the graph we can see the distribution is good, not totally biased to a single breed.

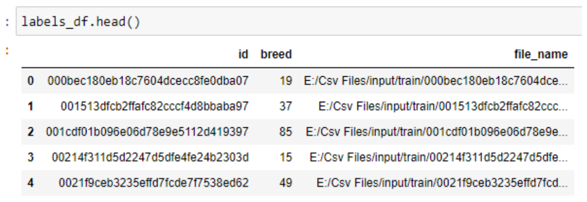
3.3 Preprocessing the dataset

Firstly we assigned a key value pair to each dog breed in the labels data frame, that is, we assigned each breed of the dog in the labels data frame to a number from 0-119 simply called as encoding each breed of the dog. And then we encoded the dog breeds in the labels data frame.



```
In [8]: class_to_num = dict(zip(dog_breeds,range(numClass)))
        class_to_num

Out[8]: {'affenpinscher': 0,
         'afghan_hound': 1,
         'african_hunting_dog': 2,
         'airedale': 3,
         'american_staffordshire_terrier': 4,
         'appenzeller': 5,
         'australian_terrier': 6,
         'basenji': 7,
         'basset': 8,
         'beagle': 9,
         'bedlington_terrier': 10,
```

Fig 7(a): Encoding each breed of the dog.



```
: labels_df.head()
```

	id	breed	file_name
0	000bec180eb18c7604dcecc8fe0dba07	19	E:/Csv Files/input/train/000bec180eb18c7604dce...
1	001513dfcb2ffafc82cccf4d8bbaba97	37	E:/Csv Files/input/train/001513dfcb2ffafc82ccc...
2	001cdf01b096e06d78e9e5112d419397	85	E:/Csv Files/input/train/001cdf01b096e06d78e9e...
3	00214f311d5d2247d5dfe4fe24b2303d	15	E:/Csv Files/input/train/00214f311d5d2247d5dfe...
4	0021f9ceb3235effd7fcde7f7538ed62	49	E:/Csv Files/input/train/0021f9ceb3235effd7fcd...

Fig 7(b): Encoding in the labels data frame.

3.4 Feature Extraction

In the feature extraction part we use the keras library which is available in python to import pre trained models. These pre-trained models are already trained so they give a lot more accurate predictions than a newly created model. For this project I have used 3 different types of pre-trained models. Those are Inception-ResNet-V2, InceptionV3, Xception.

Inception-ResNet-v2

Inception-ResNet-v2 is a convolutional neural network trained on over 1 million images from the ImageNet database. The network is 164 layers deep
|o9 gvcs`0 and can classify images into 1000 object categories such as keyboards, mice, pencils, and many animals. As a result, the network learned rich representations of a wide range of images. The input image size of the network is 299 x 299 and the output is a list of estimated class probabilities.
Based on a combination of starting structures and remaining connections. The starting ResNet block combines a multi-size convolutional filter with a residual connection. Using residual connections not only avoids the problem of degradation caused by

deep structures, but also reduces training time. This diagram shows the basic network architecture of Inception-Resnet-v2.

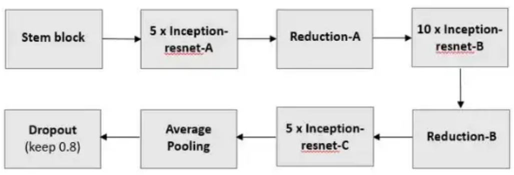


Fig 8 : Architecture of InceptionResnetV2

Inception V3

Inception v3 is an image recognition model that has been shown to attain greater than 78.1% accuracy on the ImageNet dataset. The model is the culmination of many ideas developed by multiple researchers over the years. It is based on the original paper: "Rethinking the Inception Architecture for Computer Vision" by Szegedy, et. al.The model itself is made up of symmetric and asymmetric building blocks, including convolutions, average pooling, max pooling, concatenations, dropouts, and fully connected layers. Batch normalization is used extensively throughout the model and applied to activation inputs. Loss is computed using Softmax.

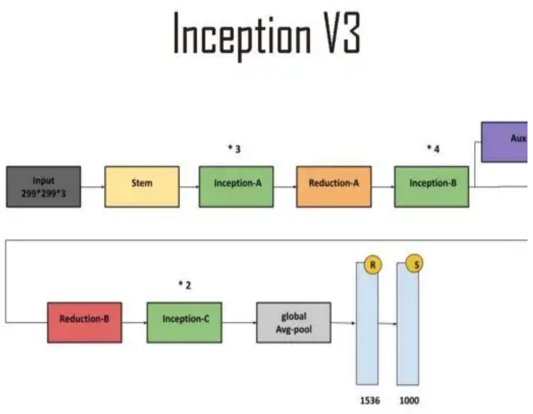


Fig 9: Inception V3 Architecture

Xception

Xception is a convolutional neural network that is 71 layers deep. You can load a pre-trained version of the network trained on more than a million images from the ImageNet database. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 299-by-299.

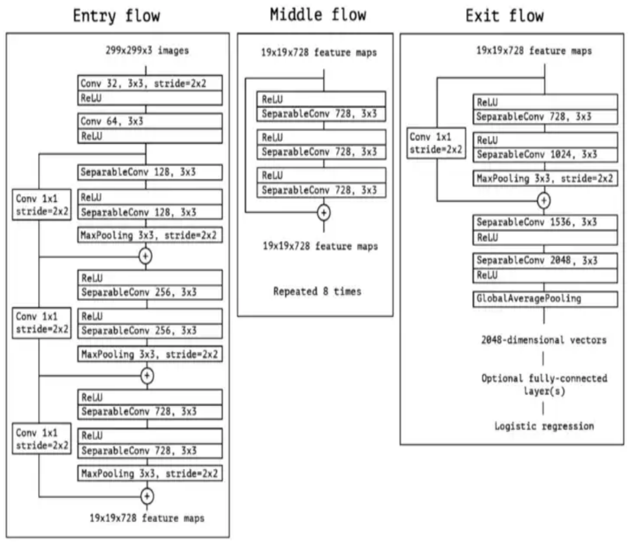


Fig 10: Architecture of Xception

After importing the models we then go to concatenate them and get the combined feature map.

3.5 Model Building

The basic idea here is we are building an architecture based on the output feature maps created from each pre-trained model and use them altogether. First I have researched on how each model performs and then seeing how they perform led me to the concept of merging of layers.

keras.layers.merge.Concatenate(axis=-1)

Layer that concatenates a list of inputs.

It takes as input a list of tensors, all of the same shape except for the concatenation axis, and returns a single tensor, the concatenation of all inputs.

In this the input will be the feature maps created by pre-trained models.

Every pre-trained model has its own way of preprocessing the input images. All of the preprocessing methods are already available in the keras library.First, we send all the images through each and every pre-trained model, they preprocess each and every image using CNN and output a feature map which consists mainly of numbers. And then after getting those feature maps we merge them using a function called Concatenate which is available in keras.

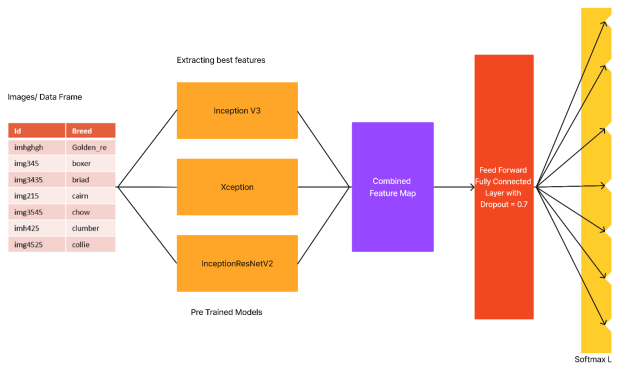
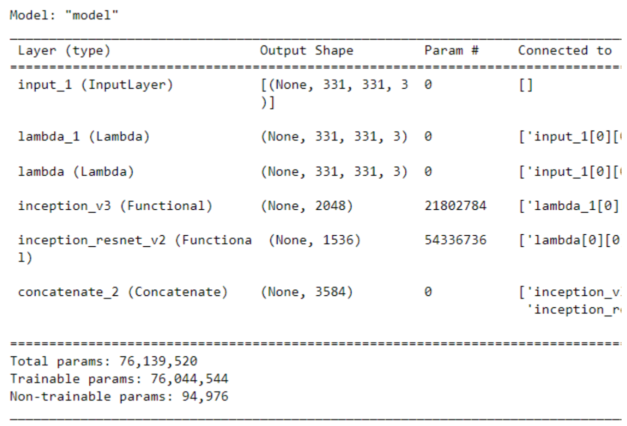


Fig 11: Model Design

In my model first I have concatenated two models. At first I checked the accuracy, then later I concatenated 3 models. I couldn't go further than 3 models due to lack of enough resources. First, I have concatenated InceptionResnetV2 and InceptionV3 and extracted features and put the features in the neural network.Then I have concatenated InceptionResnetV2 and InceptionV3 and Xception, extracted their features and put the features in the neural network. The input size of the image we are taking is (331,331,3) , 331x331 would be matrix

The variation of accuracy by concatenating 2 models and by concatenating 3 models isn't much. Even with 2 models we get really good accuracy, nearly the same as with concatenating 3 models.

Here the model means the CNN model not the fully connected layer model.



```
Model: "model"
______________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==============================================================================
 input_1 (InputLayer)           [(None, 331, 331, 3  0           []
                                )]

 lambda_1 (Lambda)              (None, 331, 331, 3)  0           ['input_1[0][

 lambda (Lambda)                (None, 331, 331, 3)  0           ['input_1[0][

 inception_v3 (Functional)      (None, 2048)         21802784    ['lambda_1[0]

 inception_resnet_v2 (Functiona  (None, 1536)        54336736    ['lambda[0][0
 l)

 concatenate_2 (Concatenate)    (None, 3584)         0           ['inception_v
                                                                  'inception_r

==============================================================================
Total params: 76,139,520
Trainable params: 76,044,544
Non-trainable params: 94,976
______________________________________________________________________________
```

Fig 12: 2 models summary

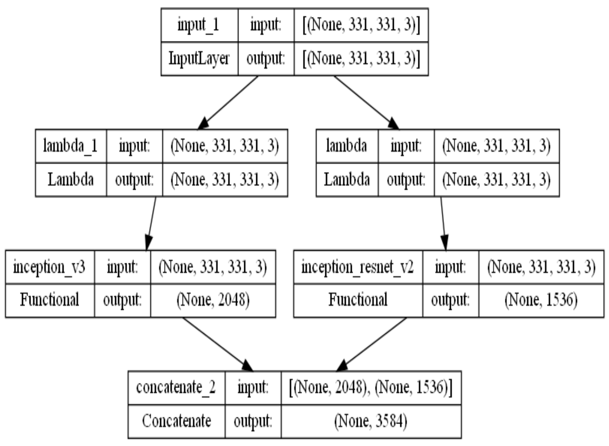
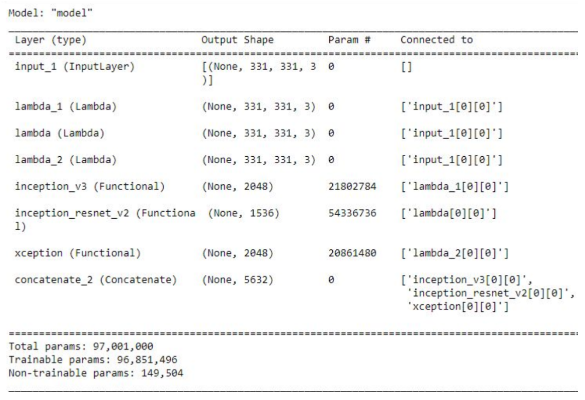


Fig 13: 2 Models plot



```
Model: "model"
______________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==============================================================================
 input_1 (InputLayer)           [(None, 331, 331, 3  0           []
                                )]

 lambda_1 (Lambda)              (None, 331, 331, 3)  0           ['input_1[0][0]']

 lambda (Lambda)                (None, 331, 331, 3)  0           ['input_1[0][0]']

 lambda_2 (Lambda)              (None, 331, 331, 3)  0           ['input_1[0][0]']

 inception_v3 (Functional)      (None, 2048)         21802784    ['lambda_1[0][0]']

 inception_resnet_v2 (Functiona  (None, 1536)        54336736    ['lambda[0][0]']
 l)

 xception (Functional)          (None, 2048)         20861480    ['lambda_2[0][0]']

 concatenate_2 (Concatenate)    (None, 5632)         0           ['inception_v3[0][0]',
                                                                  'inception_resnet_v2[0][0]',
                                                                  'xception[0][0]']

==============================================================================
Total params: 97,001,000
Trainable params: 96,851,496
Non-trainable params: 149,504
______________________________________________________________________________
```

Fig 14: 3 Models Summary

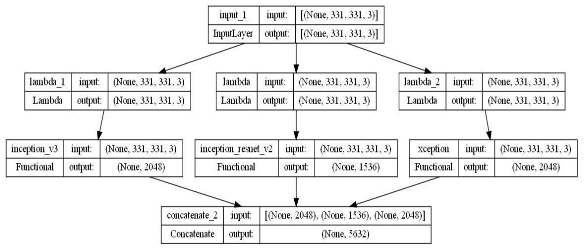
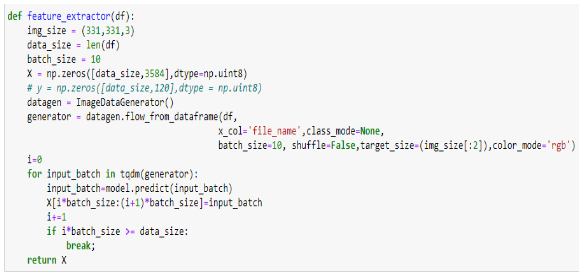


Fig 15: 3 Models Plot

In the summary we can see how many parameters the concatenate algorithm takes from each pre-trained model.

3.6 Feature Extraction

Now that the model designing is completed now comes the part of the feature extractor.



```
def feature_extractor(df):
    img_size = (331,331,3)
    data_size = len(df)
    batch_size = 10
    X = np.zeros([data_size,3584],dtype=np.uint8)
    # y = np.zeros([data_size,120],dtype = np.uint8)
    datagen = ImageDataGenerator()
    generator = datagen.flow_from_dataframe(df,
                                            x_col='file_name',class_mode=None,
                                            batch_size=10, shuffle=False,target_size=(img_size[:2]),color_mode='rgb')
    i=0
    for input_batch in tqdm(generator):
        input_batch=model.predict(input_batch)
        X[i*batch_size:(i+1)*batch_size]=input_batch
        i+=1
        if i*batch_size >= data_size:
            break;
    return X
```

Fig 16: Feature Extractor code.

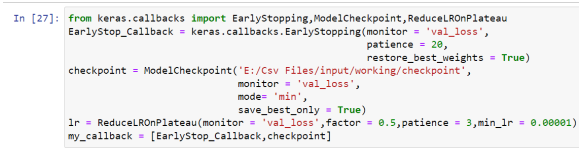
Here we took the image size of 331 × 331 as we were given the same and then we send images of batches of size 10 through ImageDataGenerator of Keras.

Keras ImageDataGenerator is used for getting the input of the original data and further, it makes the transformation of this data on a random basis and gives the output containing only the data that is newly transformed. It does not add the data. Keras image data generator class is also used to carry out data augmentation where we aim to gain the overall increment in the generalization of the model. Operations such as rotations, translations, shearing, scale changes, and horizontal flips are carried out randomly in data augmentation using an image data generator.

The output of the above code is a matrix which contains features for all the 10222 training images. This is used to compare features later in the training.

3.7 Training the model

Before starting the training we are defining callbacks. A callback is a set of functions to be applied at given stages of the training procedure. You can use callbacks to get a view on internal states and statistics of the model during training.



```
In [27]: from keras.callbacks import EarlyStopping,ModelCheckpoint,ReduceLROnPlateau
         EarlyStop_Callback = keras.callbacks.EarlyStopping(monitor = 'val_loss',
                                                            patience = 20,
                                                            restore_best_weights = True)
         checkpoint = ModelCheckpoint('E:/Csv Files/input/working/checkpoint',
                                      monitor = 'val_loss',
                                      mode= 'min',
                                      save_best_only = True)
         lr = ReduceLROnPlateau(monitor = 'val_loss',factor = 0.5,patience = 3,min_lr = 0.00001)
         my_callback = [EarlyStop_Callback,checkpoint]
```

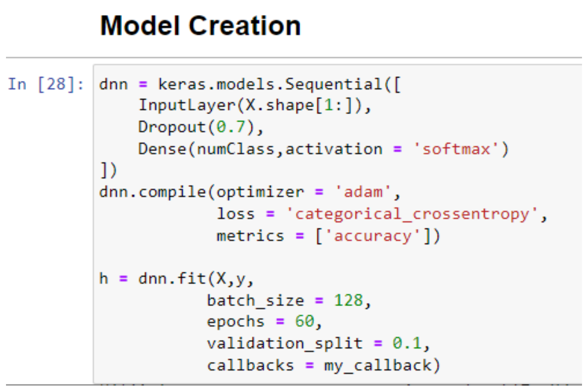
Fig 17: Callbacks Code

We are using EarlyStopping callbacks, ModemCheckpoint and ReduceLRonPlateau callbacks for our training.

We also defined a checkpoint where we save our best epochs. The learning rate method we are using is ReduceROnPlateau which decreases the learning rate when our validation loss doesn't decrease much.

For Training, we'll use a normal sequential model in which we input an array of all the extracted features which are extracted from the concatenation feature extractor.The dropout for the neural network is 0.7

And there will be a dense layer (keras DENSE) which consists of 120 neurons (all the breeds of dogs). Activation function would be SoftMax, optimizing function is Adam and loss is calculated by cross entropy method. The metric used in training is "Accuracy".



Model Creation

```
In [28]: dnn = keras.models.Sequential([
             InputLayer(X.shape[1:]),
             Dropout(0.7),
             Dense(numClass,activation = 'softmax')
         ])
         dnn.compile(optimizer = 'adam',
                     loss = 'categorical_crossentropy',
                     metrics = ['accuracy'])

         h = dnn.fit(X,y,
                    batch_size = 128,
                    epochs = 60,
                    validation_split = 0.1,
                    callbacks = my_callback)
```

Fig 18: Model Creation

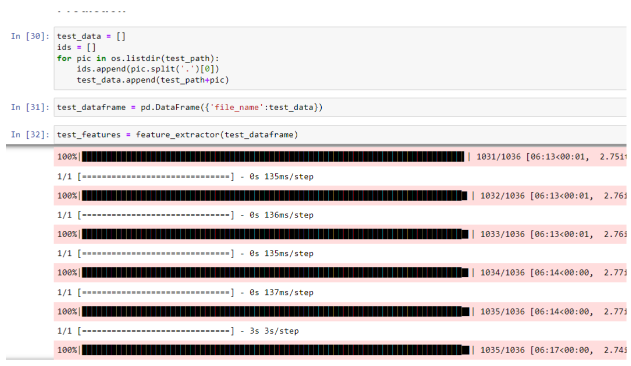
While training the model, the accuracy we got is different for the amount of pre-trained models we have used.

3.8 Prediction

For prediction we will use the test dataset which we are given. First, we append all the file names of test images into an array.

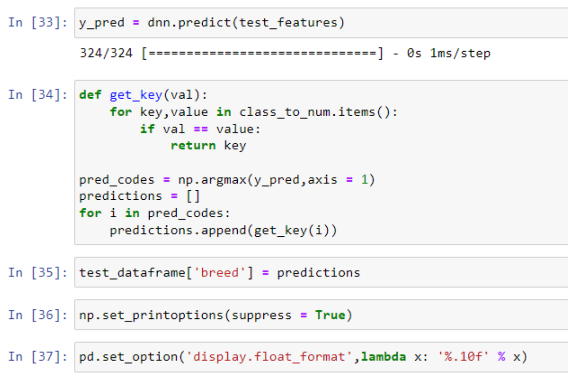
Then we create a new data frame because our feature extractor supports only data frame.We first need to take the pictures of the test data using os library and apply the feature extraction method we created earlier on them.

And then we would pass the output values of the feature extraction to the prediction method.



```
In [30]: test_data = []
         ids = []
         for pic in os.listdir(test_path):
             ids.append(pic.split('.')[0])
             test_data.append(test_path+pic)

In [31]: test_dataframe = pd.DataFrame({'file_name':test_data})

In [32]: test_features = feature_extractor(test_dataframe)

         100%|████████████████████████████████████████| 1031/1036 [06:13<00:01,  2.75i
         1/1 [==============================] - 0s 135ms/step
         100%|████████████████████████████████████████| 1032/1036 [06:13<00:01,  2.76i
         1/1 [==============================] - 0s 136ms/step
         100%|████████████████████████████████████████| 1033/1036 [06:13<00:01,  2.76i
         1/1 [==============================] - 0s 135ms/step
         100%|████████████████████████████████████████| 1034/1036 [06:14<00:00,  2.77i
         1/1 [==============================] - 0s 137ms/step
         100%|████████████████████████████████████████| 1035/1036 [06:14<00:00,  2.77i
         1/1 [==============================] - 3s 3s/step
         100%|████████████████████████████████████████| 1035/1036 [06:17<00:00,  2.74i
```

Fig 19 (a): Prediction(Extracting features on test data set)



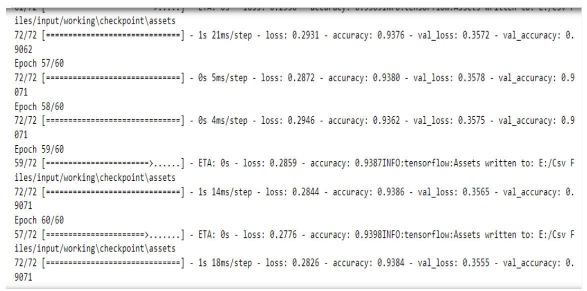
```
In [33]: y_pred = dnn.predict(test_features)

         324/324 [==============================] - 0s 1ms/step

In [34]: def get_key(val):
             for key,value in class_to_num.items():
                 if val == value:
                     return key

         pred_codes = np.argmax(y_pred,axis = 1)
         predictions = []
         for i in pred_codes:
             predictions.append(get_key(i))

In [35]: test_dataframe['breed'] = predictions

In [36]: np.set_printoptions(suppress = True)

In [37]: pd.set_option('display.float_format',lambda x: '%.10f' % x)
```

Fig 19 (b): Prediction(predicting the values)

4. Results

Initially when we saw some other works implementing using a single pre-trained model they gave a good accuracy of 89% but we felt we can do better than that. Then we tried merging 2 or more layers of pre-trained models pre-processed input. So when we went with multi layers we got the following results.

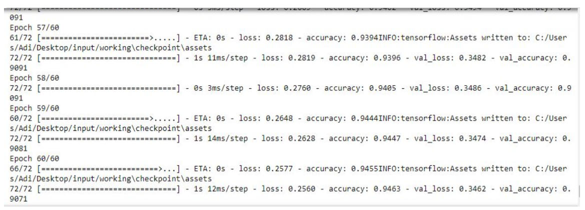
When we used 2 pre-trained models (inceptionV3 and inceptionResNet-v2) we got an accuracy of 93.8 - 93.9 we can round it off to 94%, also the validation loss is 0.355 with validation accuracy of 90% on average.



```
iles/input/working\checkpoint\assets
72/72 [==============================] - 1s 21ms/step - loss: 0.2931 - accuracy: 0.9376 - val_loss: 0.3572 - val_accuracy: 0.
9062
Epoch 57/60
72/72 [==============================] - 0s 5ms/step - loss: 0.2872 - accuracy: 0.9380 - val_loss: 0.3578 - val_accuracy: 0.9
071
Epoch 58/60
72/72 [==============================] - 0s 4ms/step - loss: 0.2946 - accuracy: 0.9362 - val_loss: 0.3575 - val_accuracy: 0.9
071
Epoch 59/60
59/72 [=======================>......] - ETA: 0s - loss: 0.2859 - accuracy: 0.9387INFO:tensorflow:Assets written to: E:/Csv F
iles/input/working\checkpoint\assets
72/72 [==============================] - 1s 14ms/step - loss: 0.2844 - accuracy: 0.9386 - val_loss: 0.3565 - val_accuracy: 0.
9071
Epoch 60/60
57/72 [=====================>.......] - ETA: 0s - loss: 0.2776 - accuracy: 0.9398INFO:tensorflow:Assets written to: E:/Csv F
iles/input/working\checkpoint\assets
72/72 [==============================] - 1s 18ms/step - loss: 0.2826 - accuracy: 0.9384 - val_loss: 0.3555 - val_accuracy: 0.
9071
```

Fig 20: Accuracy of 2 Pre Trained models.

And when we used 3 pre-trained models (inception v3, inceptionResNet v2 and Xception) we got accuracy on an average of 94.5% not much difference with the 2 pre-trained models. Also the validation loss is 0.34 and validation accuracy is the same that is 90%



```
091
Epoch 57/60
61/72 [========================>.....] - ETA: 0s - loss: 0.2818 - accuracy: 0.9394INFO:tensorflow:Assets written to: C:/User
s/Adi/Desktop/input/working\checkpoint\assets
72/72 [==============================] - 1s 11ms/step - loss: 0.2819 - accuracy: 0.9396 - val_loss: 0.3482 - val_accuracy: 0.
9091
Epoch 58/60
72/72 [==============================] - 0s 3ms/step - loss: 0.2760 - accuracy: 0.9405 - val_loss: 0.3486 - val_accuracy: 0.9
091
Epoch 59/60
60/72 [========================>.....] - ETA: 0s - loss: 0.2648 - accuracy: 0.9444INFO:tensorflow:Assets written to: C:/User
s/Adi/Desktop/input/working\checkpoint\assets
72/72 [==============================] - 1s 14ms/step - loss: 0.2628 - accuracy: 0.9447 - val_loss: 0.3474 - val_accuracy: 0.
9081
Epoch 60/60
66/72 [==========================>...] - ETA: 0s - loss: 0.2577 - accuracy: 0.9455INFO:tensorflow:Assets written to: C:/User
s/Adi/Desktop/input/working\checkpoint\assets
72/72 [==============================] - 1s 12ms/step - loss: 0.2560 - accuracy: 0.9463 - val_loss: 0.3462 - val_accuracy: 0.
9071
```

Fig 21: Accuracy of 3 Pre Trained models.

Accumulating 2 or more pre-trained models was a good idea, as it gave a lot of difference when tried with a single pre-trained model.

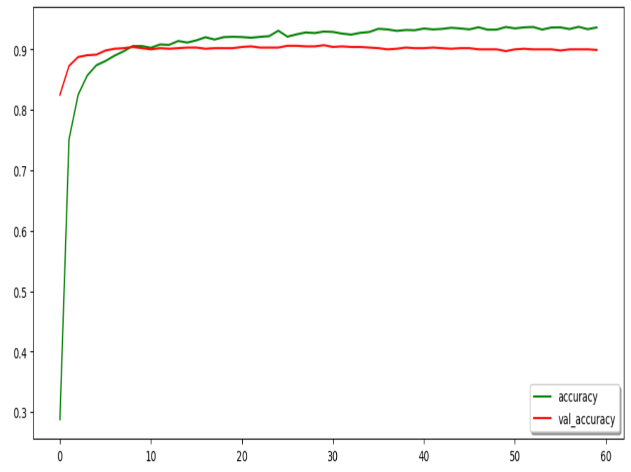
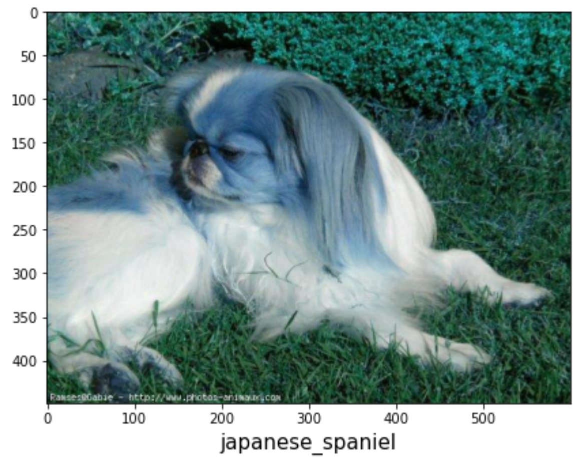
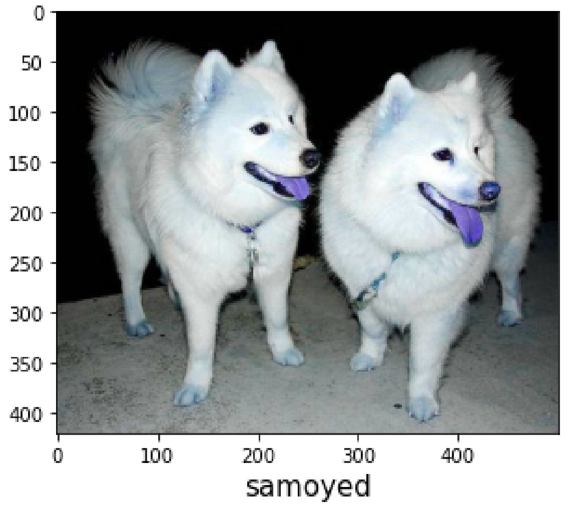


Fig 22 : Accuracy and Validation Accuracy graph throughout the epochs.

Below are some of the predictions done by the model from the test data.





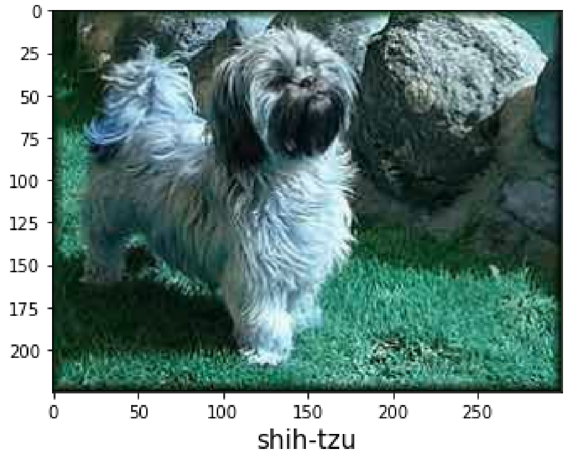


Fig 23: Model predicting the dog breed names.

S No	Models Used	Accuracy Obtained
1.	Inception V3 Inception ResNet-v2	93.4 - 94%
2.	Inception V3, Inception ResNet - v2, Xception	94.7-95%

Table 2: Accuracy Table for our model

5. Conclusion and Future Scope

The motivation of this model was to learn how to use a machine learning classification tool in order to classify images, namely, dog breeds. In this paper we have explained how to build a dog breed classification using pre-trained models. For further improvising it we can also deploy the model in some android application or a web application and we can use it in our mobile phones.

In future research, we would like to improve the prediction accuracy rate. After doing the above project we have learned that CNN is majorly used for Image recognition using given images as a dataset. This work also implies that CNN can be used for various types of classifications not only dog breeds. The similar model can be implemented for the distinction of plants, humans, cats and other animals. Also in order to improve the current version, we can go for different data sets with a lot more images and better images.

6. References

An Intelligent Dog Breed Recognition System Using Deep Learning | International Journal of Data Informatics and Intelligent Computing (ijdiic.com)

https://www.reev.us/pdfs/mulligan2019dog.pdf

Dog Breed Identification Using Convolutional Neural Networks on Android | CVR Journal of Science and Technology

Breakthrough Conventional Based Approach for Dog Breed Classification Using CNN with Transfer Learning | IEEE Conference Publication | IEEE Xplore

Knowing Your Dog Breed: Identifying a Dog Breed with Deep Learning | SpringerLink

CVPR 2020 Open Access Repository (thecvf.com)