Machine Learning Advance Nanodegree

Capstone Project Proposal: DOG BREED CLASSIFICATION

Mehak Piplani

April 17, 2018

PROJECT OVERVIEW

This project belongs to the domain of Image recognition. Image recognition has gained a lot of interest more recently which is driven by the demand for more sophisticated algorithms and advances in processing capacity of the computation devices. These algorithms have been integrated in our daily life from cell phone passwords to car-plate recognition. Normal classiﬁcation of objects has become common and I really got interested by the idea of the kaggle competition regarding Dog breed classiﬁcation

The objective of this project is to train the model which can predict the breeds of dog based on the image given as input. The dataset used is the Imagenet database in the pre trained models of keras and to enhance and augment the dataset I have also used the data set provided by Udacity which contains images corresponding to the 133 breeds of dogs.

I present a compare and contrast between the various models used.

PROBLEM STATEMENT:

The problem of identifying the breed of the dog from an image is a type of problem where a machine if trained well outperforms the human power of classification. A normal person can identify and differentiate between max 4-5 breeds of dogs and this problem becomes widened if this problem extends to classifying the breeds of other animals like monkeys, cats, etc.

From the above problem statement , it can be clearly seen that this is a dense classiﬁcation problem where many types of minute features are extracted from the image for identiﬁcation.

Once trained the model is able to predict the breed from an unseen image based on the concepts of transfer learning and convolutional neural networks. The models used are from the keras framework include vgg16, resnet50 and inception. The models are used as feature extractor and applied on the dataset given by the Udacity team available here:

METRICS:

The output of the model in this problem will be based on the fact which breed seems to have more probability for that particular image. Since these probabilities are of mutually exclusive classes (a dog can belong to one breed)and hence these sum up to 1 .

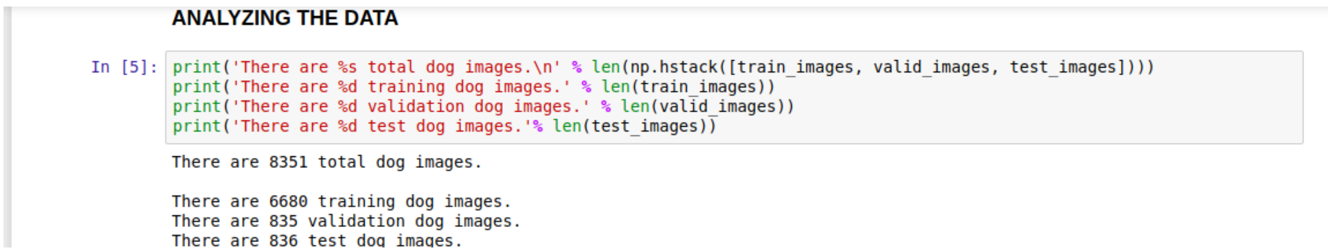
For this problem I have considered to separate metrics:

1) Accuracy on the test set
   This has been compared and contrasted between the various models used.
   This is implemented by comparing the predicted labels to the values of already available labels and a score is calculated by counting the number of instances which match.

2) Multi class log loss
   This is the function chosen by kaggle .For a multiclass classification problem like the one described above log loss tells us about the confidence of the model. The main objective would be minimize this log loss value and the value of 0 would indicate a perfect model.

DATA EXLORATION AND ANALYSIS

1) This displays the total number of images in the training , validation and testing data set.
   So basically along with the imagenet dataset described below the total number of images of the augmented dataset includes 8351 images.

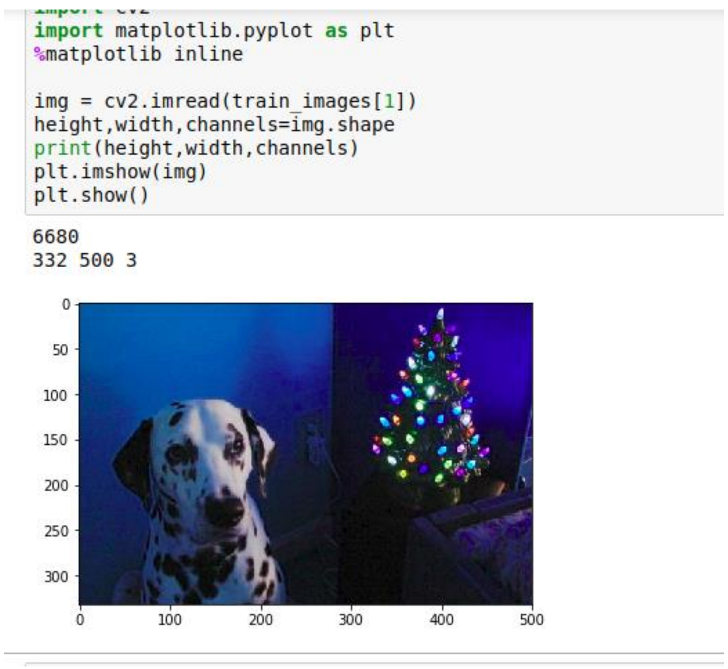


ANALYZING THE DATA

```
In [5]: print('There are %s total dog images.\n' % len(np.hstack([train_images, valid_images, test_images])))
        print('There are %d training dog images.' % len(train_images))
        print('There are %d validation dog images.' % len(valid_images))
        print('There are %d test dog images.'% len(test_images))
```

```
There are 8351 total dog images.

There are 6680 training dog images.
There are 835 validation dog images.
There are 836 test dog images.
```

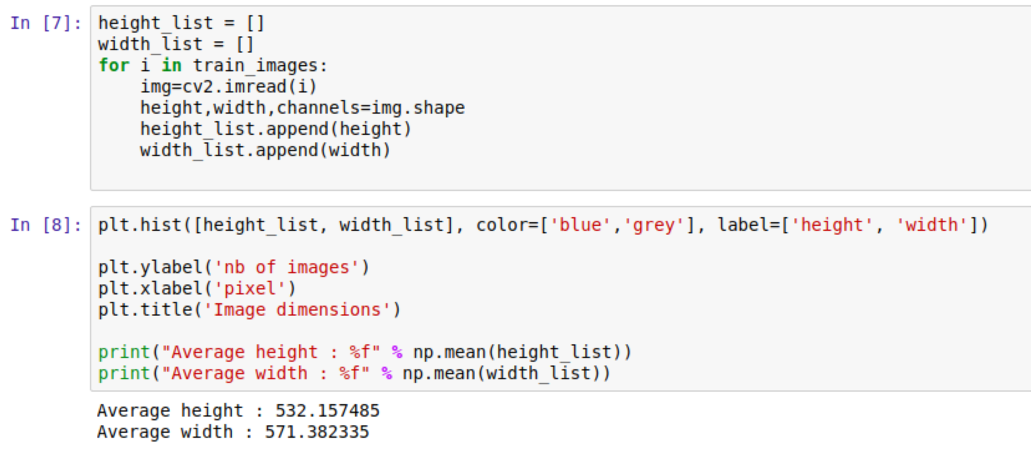
2) This shows an example of image displaying the type of images and image dimensions
   From this we infer that the images consist of not only dogs, it also contains other objects in the images. The dimensions of image consist of image height, width and size of channel =3 (RGB)



```
import cv2
import matplotlib.pyplot as plt
%matplotlib inline

img = cv2.imread(train_images[1])
height,width,channels=img.shape
print(height,width,channels)
plt.imshow(img)
plt.show()
```

```
6680
332 500 3
```

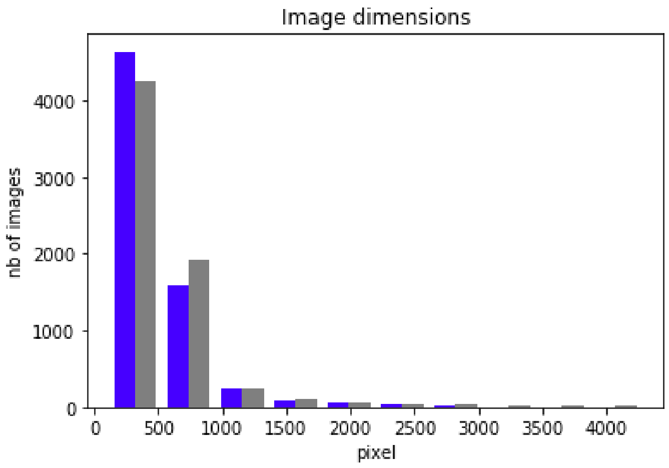
3) Then all image sizes have been observed and on an average these images have an average height of 532 and width of 571. A plot is also shown showing the variations of dimensions corresponding to the number of pixels in an image.



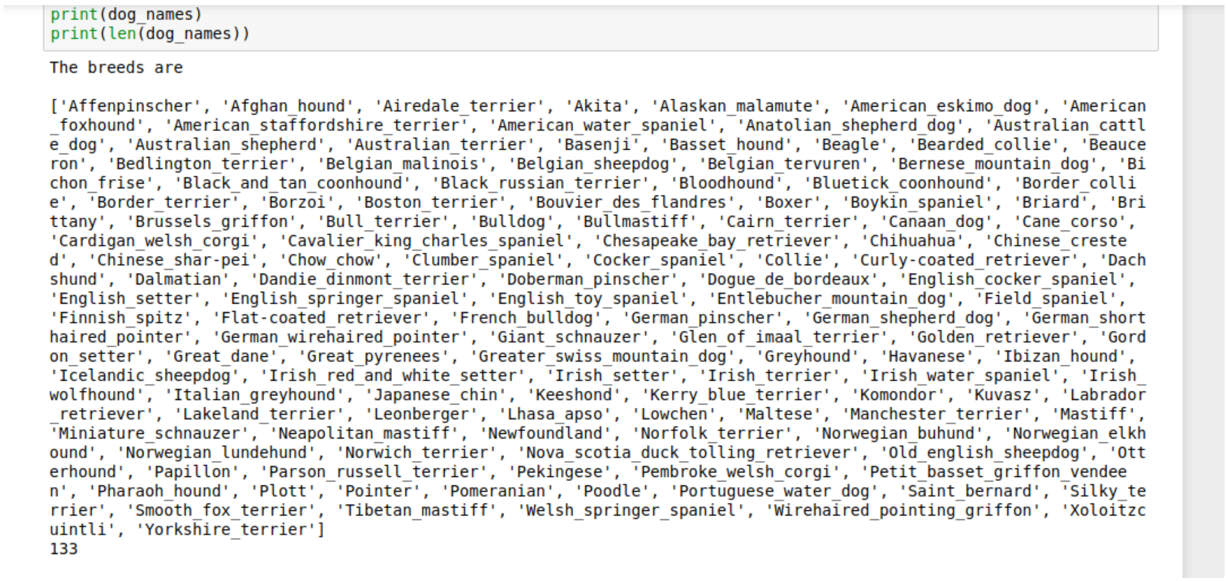
```
In [7]: height_list = []
        width_list = []
        for i in train_images:
            img=cv2.imread(i)
            height,width,channels=img.shape
            height_list.append(height)
            width_list.append(width)
```

```
In [8]: plt.hist([height_list, width_list], color=['blue','grey'], label=['height', 'width'])

        plt.ylabel('nb of images')
        plt.xlabel('pixel')
        plt.title('Image dimensions')

        print("Average height : %f" % np.mean(height_list))
        print("Average width : %f" % np.mean(width_list))
```

```
Average height : 532.157485
Average width : 571.382335
```



4) This is to inquire regarding the various breeds i.e. their names



```
print(dog_names)
print(len(dog_names))
```

```
The breeds are

['Affenpinscher', 'Afghan_hound', 'Airedale_terrier', 'Akita', 'Alaskan_malamute', 'American_eskimo_dog', 'American
_foxhound', 'American_staffordshire_terrier', 'American_water_spaniel', 'Anatolian_shepherd_dog', 'Australian_cattl
e_dog', 'Australian_shepherd', 'Australian_terrier', 'Basenji', 'Basset_hound', 'Beagle', 'Bearded_collie', 'Beauce
ron', 'Bedlington_terrier', 'Belgian_malinois', 'Belgian_sheepdog', 'Belgian_tervuren', 'Bernese_mountain_dog', 'Bi
chon_frise', 'Black_and_tan_coonhound', 'Black_russian_terrier', 'Bloodhound', 'Bluetick_coonhound', 'Border_colli
e', 'Border_terrier', 'Borzoi', 'Boston_terrier', 'Bouvier_des_flandres', 'Boxer', 'Boykin_spaniel', 'Briard', 'Bri
ttany', 'Brussels_griffon', 'Bull_terrier', 'Bulldog', 'Bullmastiff', 'Cairn_terrier', 'Canaan_dog', 'Cane_corso',
'Cardigan_welsh_corgi', 'Cavalier_king_charles_spaniel', 'Chesapeake_bay_retriever', 'Chihuahua', 'Chinese_creste
d', 'Chinese_shar-pei', 'Chow_chow', 'Clumber_spaniel', 'Cocker_spaniel', 'Collie', 'Curly-coated_retriever', 'Dach
shund', 'Dalmatian', 'Dandie_dinmont_terrier', 'Doberman_pinscher', 'Dogue_de_bordeaux', 'English_cocker_spaniel',
'English_setter', 'English_springer_spaniel', 'English_toy_spaniel', 'Entlebucher_mountain_dog', 'Field_spaniel',
'Finnish_spitz', 'Flat-coated_retriever', 'French_bulldog', 'German_pinscher', 'German_shepherd_dog', 'German_short
haired_pointer', 'German_wirehaired_pointer', 'Giant_schnauzer', 'Glen_of_imaal_terrier', 'Golden_retriever', 'Gord
on_setter', 'Great_dane', 'Great_pyrenees', 'Greater_swiss_mountain_dog', 'Greyhound', 'Havanese', 'Ibizan_hound',
'Icelandic_sheepdog', 'Irish_red_and_white_setter', 'Irish_setter', 'Irish_terrier', 'Irish_water_spaniel', 'Irish_
wolfhound', 'Italian_greyhound', 'Japanese_chin', 'Keeshond', 'Kerry_blue_terrier', 'Komondor', 'Kuvasz', 'Labrador
_retriever', 'Lakeland_terrier', 'Leonberger', 'Lhasa_apso', 'Lowchen', 'Maltese', 'Manchester_terrier', 'Mastiff',
'Miniature_schnauzer', 'Neapolitan_mastiff', 'Newfoundland', 'Norfolk_terrier', 'Norwegian_buhund', 'Norwegian_elkh
ound', 'Norwegian_lundehund', 'Norwich_terrier', 'Nova_scotia_duck_tolling_retriever', 'Old_english_sheepdog', 'Ott
erhound', 'Papillon', 'Parson_russell_terrier', 'Pekingese', 'Pembroke_welsh_corgi', 'Petit_basset_griffon_vendee
n', 'Pharaoh_hound', 'Plott', 'Pointer', 'Pomeranian', 'Poodle', 'Portuguese_water_dog', 'Saint_bernard', 'Silky_te
rrier', 'Smooth_fox_terrier', 'Tibetan_mastiff', 'Welsh_springer_spaniel', 'Wirehaired_pointing_griffon', 'Xoloitzc
uintli', 'Yorkshire_terrier']
133
```

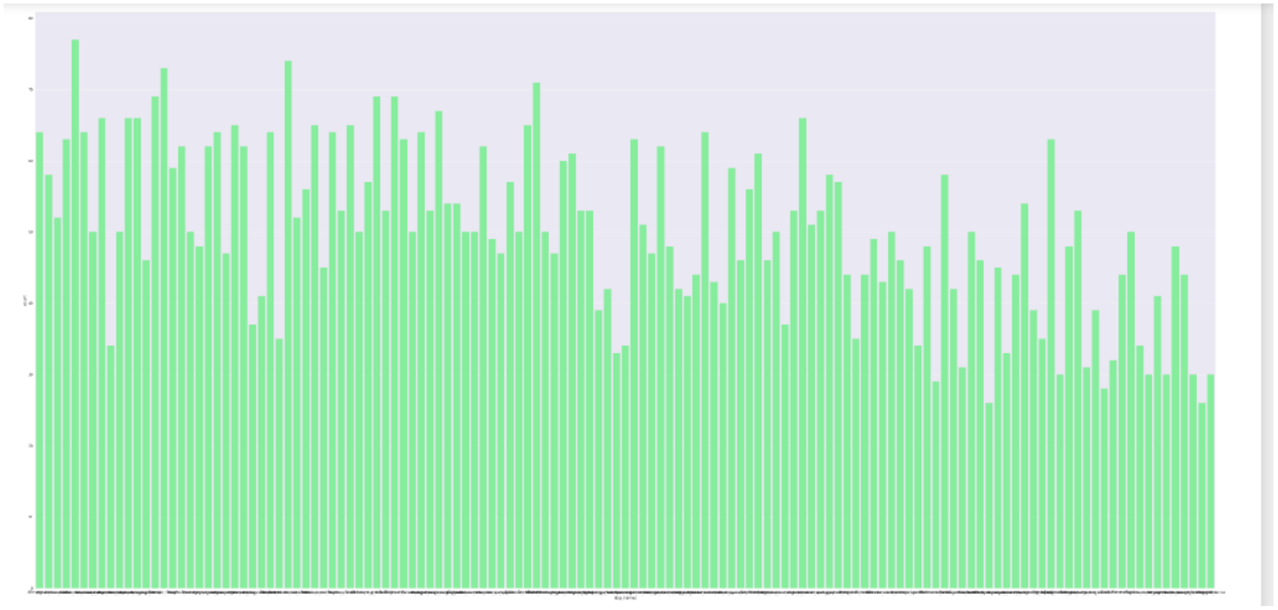
5) This is to count the number of images for a particular breed at an average

Below the plot shows the number of images for all the breeds.

```
In [10]: import seaborn as sns
         count = []
         number = 0;
         for item in sorted(glob("dogImages/train/*/")):

             for i in glob(item+'*'):
                 number=number+1
             count.append(number)
             number = 0
         print(count)
```

```
[64, 58, 52, 63, 77, 64, 50, 66, 34, 50, 66, 66, 46, 69, 73, 59, 62, 50, 48, 62, 64, 47, 65, 62, 37, 41, 64, 35, 7
4, 52, 56, 65, 45, 64, 53, 65, 50, 57, 69, 53, 69, 63, 50, 64, 53, 67, 54, 54, 50, 50, 62, 49, 47, 57, 50, 65, 71,
50, 47, 60, 61, 53, 53, 39, 42, 33, 34, 63, 51, 47, 62, 48, 42, 41, 44, 64, 43, 40, 59, 46, 56, 61, 46, 50, 37, 53,
66, 51, 53, 58, 57, 44, 35, 44, 49, 43, 50, 46, 42, 34, 48, 29, 58, 42, 31, 50, 46, 26, 45, 33, 44, 54, 39, 35, 63,
30, 48, 53, 31, 39, 28, 32, 44, 50, 34, 30, 41, 30, 48, 44, 30, 26, 30]
```

```
In [11]: sns.set(color_codes=True)
         plt.rcParams['figure.figsize'] = (60.0, 30.0)
         sns.barplot(dog_names, count, color="lightgreen")
         plt.xlabel("dog_names")
         plt.ylabel("count")
```



ALGORITHMS:

1) Convolutional Neural Networks

Convolutional Neural Networks are very similar to ordinary Neural Networks which are made up of neurons that have weights and biases which are learned during the training process. Each neuron receives some inputs, performs a dot product and optionally follows it with a non−linear activation

function. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function on the last (fully-connected) layer.
There are three main types of layers to build ConvNet architectures: Convolutional Layer, Pooling Layer, and Fully-Connected Layer and stack these layers to form a full ConvNet architecture.

CONVOLUTIONAL LAYER:
This layer functions as a type of filter applied to the input image. This layer will have the same depth as that of the input layer. This can be considered as a filter which multiplies and filter values which the pixels of the image in the form of patches and then sum of these values is calculated. This is done over the full image in the form of sliding technique with overlapping regions so we see that mainly it scans the image and finds out the specific patterns in the image.

POOLING LAYER:
This layer takes up the feature map output from the convolutional layer and prepares a more deep map by applying max pool function to the area specified like for example if its 2,2 then the square formed in the image of size 2*2, this layer extracts the maximum value from the 4 blocks.
Another approach include average pooling in which instead of the max value the average of the four values is extracted.

FULLY-CONNECTED LAYER:
This is the layer which can take input from either the convolutional or the pooling layer and then basically outputs a

vector having the dimension same as the number of classes in that problem.
This layer is mostly used as the last layers in the network and take as input the output containing the high level features and predicts a probability corresponding to the class based on the features.

The last layer that is applied is the softmax layer which is generally used in the cases of classification problems because it takes as input a vector and outputs the normalized type of vector hence giving the exact values of probabilities for the various classes.

Since this network is mostly applied in the case of images we work on an image by taking patches from an image and then performing convolution on them to extract the features and so we are able to extract really small features

2) Imagenet

ImageNet is a project of oxford which is basically aimed at labelling and categorizing images into almost 22,000 separate object categories for the purpose of research in the field of computer vision research.

However this tern in the field of deep learning refers to the ImageNet Large Scale Visual Recognition Challenge. The goal of this image classification challenge is to train the model that can correctly classify an input image upto 1000 separate categories. These 1,000 image categories represent object classes that we encounter in our day-to-day lives, such as species of dogs, cats, various household objects, vehicle types, and much more.

3) **Transfer Learning**

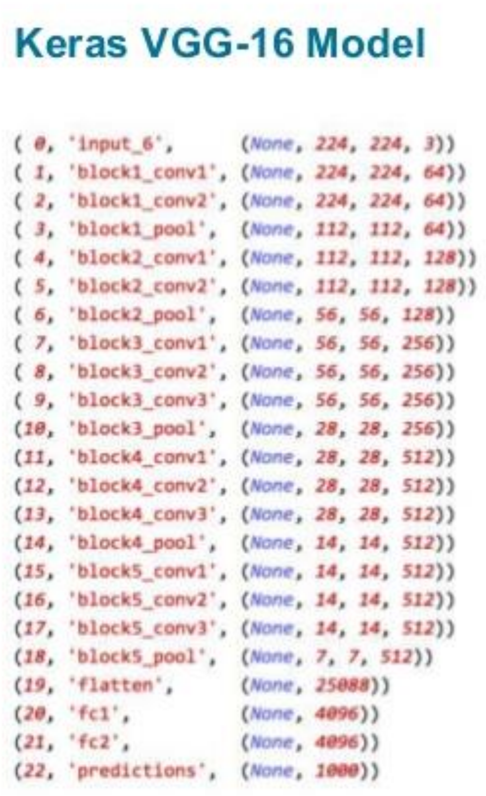
Transfer Learning using pre trained Models in Keras Transfer learning is a research problem in machine learning that focuses on storing knowledge gained from a pre trained model while solving one problem and applying it to a different but related problem. Transfer learning helps because in case of images deep networks try to detect edges in the earlier layers, Shapes in the middle layer and some high level data specific features in the later layers so using a model which has already extracted these would help and decrease the time required for computation. This algorithm would be used in two ways:

- **Feature extraction:**

In this a pre-trained model as a feature extraction mechanism. What is done is o the output layer is removed and then entire network is used as a fixed feature extractor for the new data set

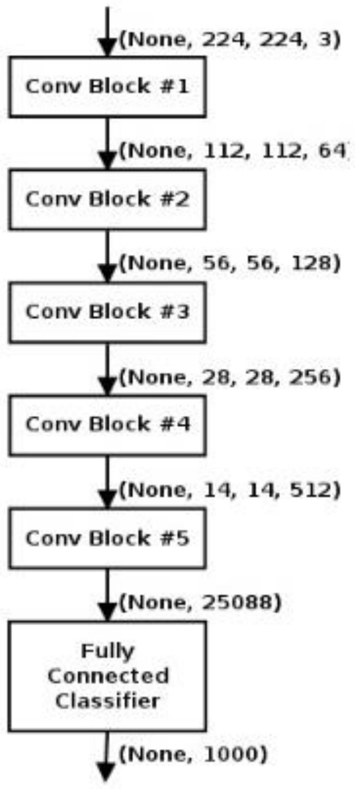
The three types of models used are:

1) VGG 16:



Keras VGG-16 Model

```
( 0, 'input_6',       (None, 224, 224, 3))
( 1, 'block1_conv1', (None, 224, 224, 64))
( 2, 'block1_conv2', (None, 224, 224, 64))
( 3, 'block1_pool',  (None, 112, 112, 64))
( 4, 'block2_conv1', (None, 112, 112, 128))
( 5, 'block2_conv2', (None, 112, 112, 128))
( 6, 'block2_pool',  (None, 56, 56, 128))
( 7, 'block3_conv1', (None, 56, 56, 256))
( 8, 'block3_conv2', (None, 56, 56, 256))
( 9, 'block3_conv3', (None, 56, 56, 256))
(10, 'block3_pool',  (None, 28, 28, 256))
(11, 'block4_conv1', (None, 28, 28, 512))
(12, 'block4_conv2', (None, 28, 28, 512))
(13, 'block4_conv3', (None, 28, 28, 512))
(14, 'block4_pool',  (None, 14, 14, 512))
(15, 'block5_conv1', (None, 14, 14, 512))
(16, 'block5_conv2', (None, 14, 14, 512))
(17, 'block5_conv3', (None, 14, 14, 512))
(18, 'block5_pool',  (None, 7, 7, 512))
(19, 'flatten',      (None, 25088))
(20, 'fc1',          (None, 4096))
(21, 'fc2',          (None, 4096))
(22, 'predictions',  (None, 1000))
```



Application:

- Given image → find object name in the image
- It can detect any one of 1000 images
- It takes input image of size 224 * 244 * 3 (RGB image)

Built using:

- Convolutions layers (used only 3*3 size)
- Max pooling layers (used only 2*2 size)
- Fully connected layers at end
- Total 16 layers

2) RESNET 50

Resnet is basically a short name for Residual Network. This network introduces the concept of residual learning which can be explained as follows:

In general in a deep convolutional neural network there are several layers which are stacked and are trained to the task in hand and these layers extract low, mid and high level features through these layers .In this instead of learning new features , we learn from the residual and Resnet does this by directly connecting the input of nth layer to some (n+i)th layer .

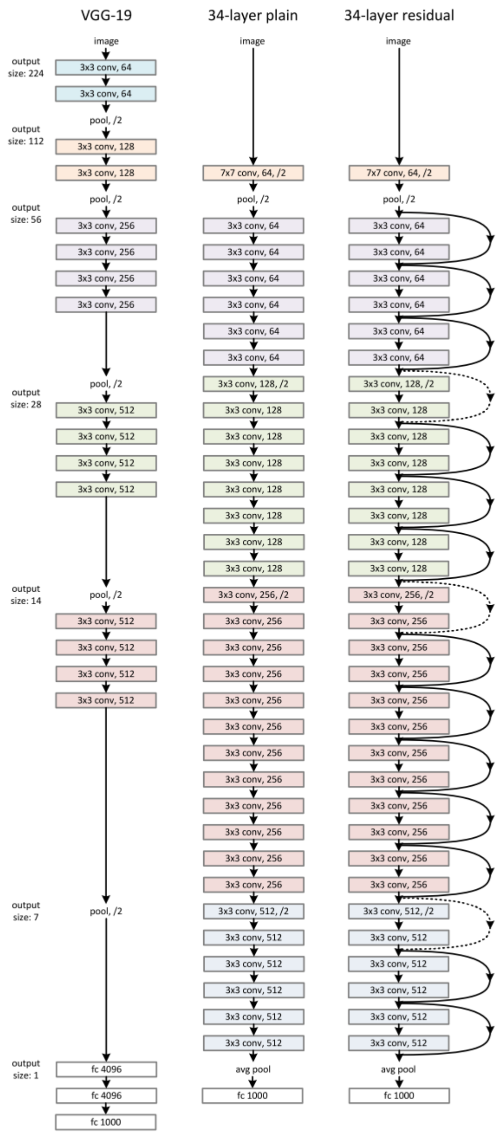
ResNet 50 is a 50 layer Residual network extended to a vgg 19 network which is variant of vgg16 network.

Application:

- Given image → find object name in the image
- It can detect any one of 1000 images
- It takes input image of size 224 * 244 * 3 (RGB image) (not smaller than 197)

Built using:

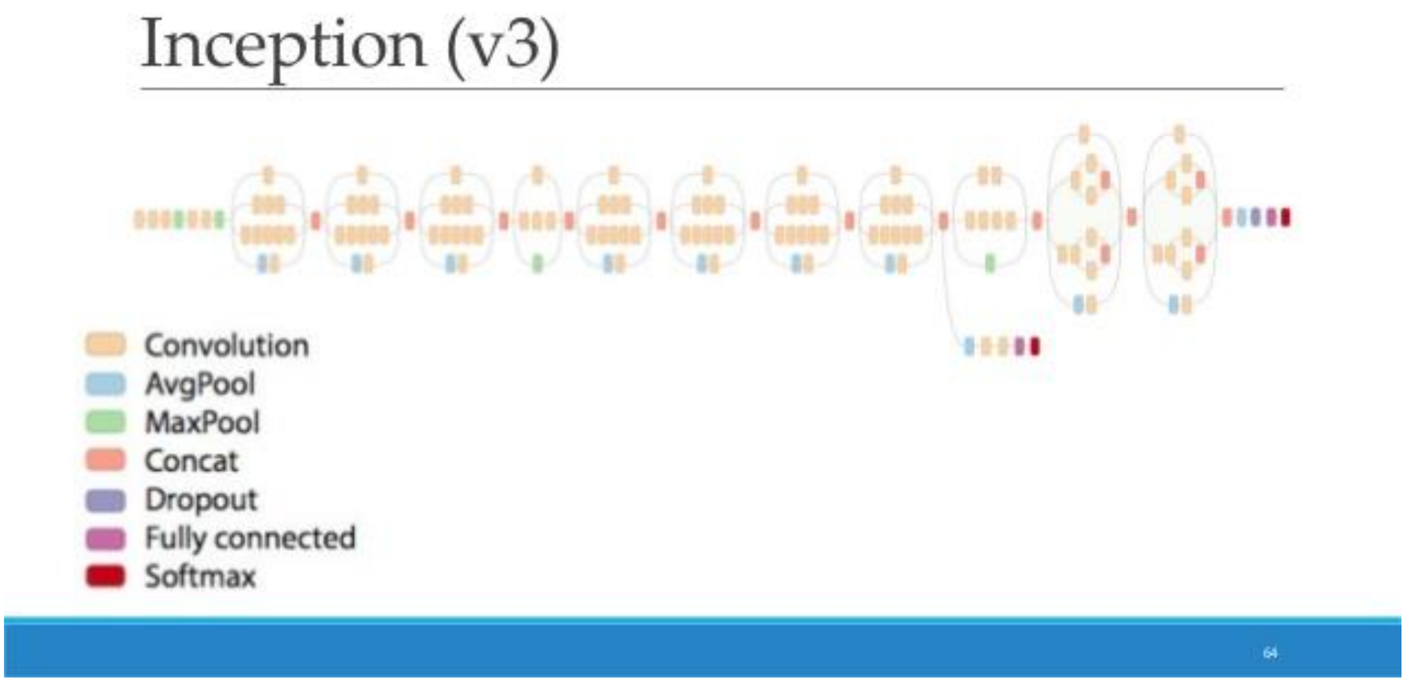
- Convolutions layers (used only 3*3 size)
- Max pooling layers (used only 2*2 size)
- Fully connected layers at end
- Total 50 layers



3) INECPTION V3

Inception V1 is a modification of GoogleNet produced by involving convolutions , layers of 5 * 5 replaced by 3*3 with up to 128 filters ,then came the inception v2 In which the concept of factorization was introduced , factorizing 7 * 7 into 3 *3 .

Then in inception v3 additional BN−auxiliary was added which refers to the fully connected layer of the auxiliary classifier is also normalized.



BENCHMARK

The benchmark is provided by the kaggle leaderboard itself and the loss value was given about 0.3 and this value was based on the model which used k−Nearest Neighbors.

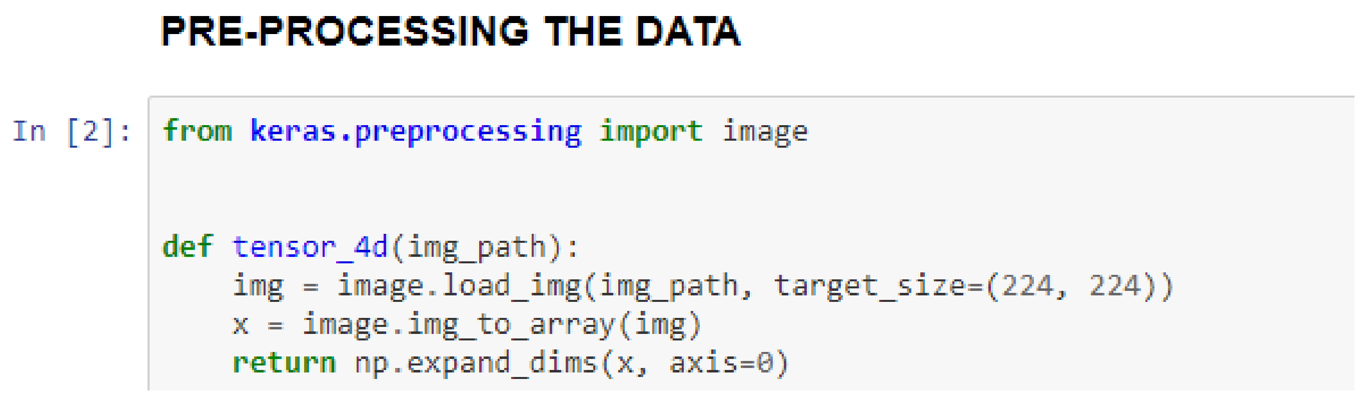
Data Pre−Processing

The first step of the project would be to pre−process the images in the testing set. This project would be done using Keras in which CNNs require a 4D array as input, with shape

(nbsamples,rows,columns,channels),

where nb samples corresponds to the total number of images (or samples), and rows, columns, and channels correspond to the number of rows, columns, and channels for each image, respectively. After converting these to tensor arrays these images would then be rescaled. Then the next step involves Getting the 4D tensor ready for for the pre−trained model in Keras, in which first the RGB image is converted to BGR by reordering the channels and then these are normalised by subtracting the mean value from the tensors.

To this I have built my functions and I am using preprocess_input method of the keras.application.vgg16 class when vgg16 model used and for others similar thing has been done.



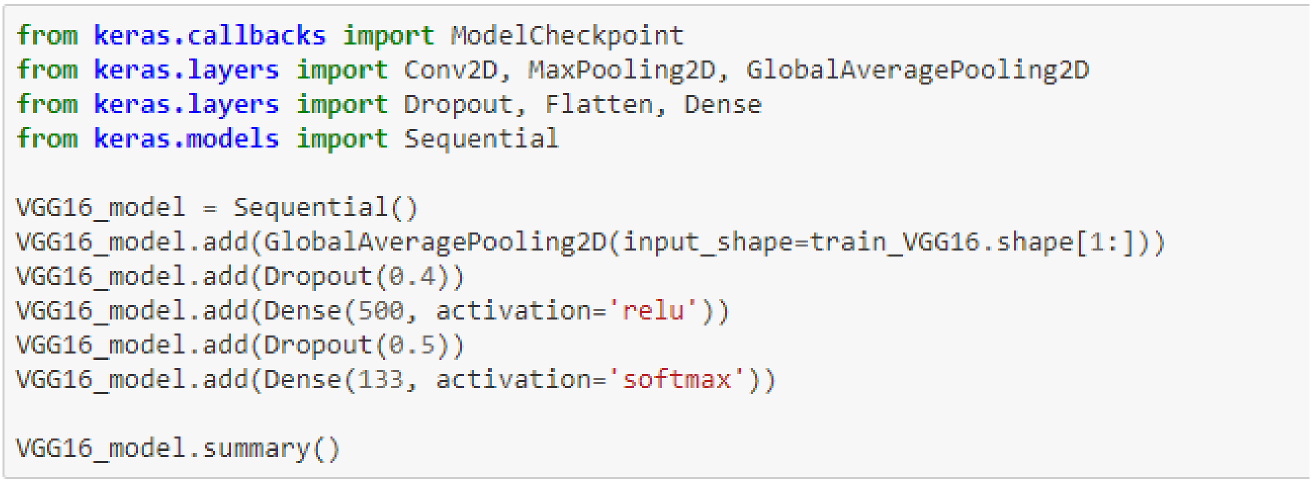
PRE-PROCESSING THE DATA

```
In [2]: from keras.preprocessing import image


        def tensor_4d(img_path):
            img = image.load_img(img_path, target_size=(224, 224))
            x = image.img_to_array(img)
            return np.expand_dims(x, axis=0)
```

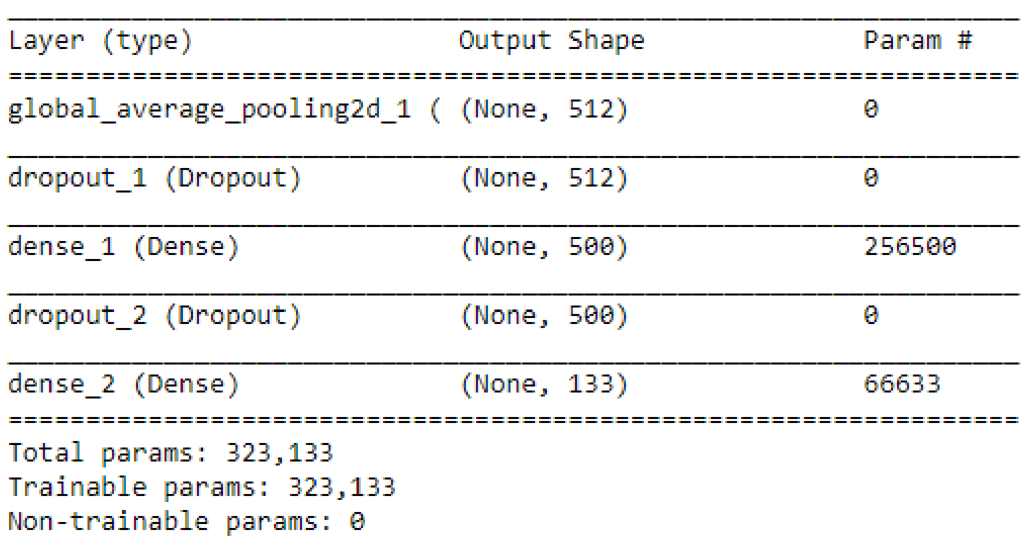
IMPLEMENTATION

I have used keras inbuilt functions using the library keras.applications and I have done this is two steps:

1) I have tried to use the model as it is and displayed on 200 images of training and testing set that whether the output is correct or not
2) I have used these models as feature extractors added Few layers of my own that then tried to predict the breed.



```
from keras.callbacks import ModelCheckpoint
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D
from keras.layers import Dropout, Flatten, Dense
from keras.models import Sequential

VGG16_model = Sequential()
VGG16_model.add(GlobalAveragePooling2D(input_shape=train_VGG16.shape[1:]))
VGG16_model.add(Dropout(0.4))
VGG16_model.add(Dense(500, activation='relu'))
VGG16_model.add(Dropout(0.5))
VGG16_model.add(Dense(133, activation='softmax'))

VGG16_model.summary()
```



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d_1 ( (None, 512)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 500)               256500
_________________________________________________________________
dropout_2 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_2 (Dense)              (None, 133)               66633
=================================================================
Total params: 323,133
Trainable params: 323,133
Non-trainable params: 0
```

RESULTS:

I tried to work with three models:

model	accuracy
VGG16	63.28%
RESNET 50	79.55%
INCEPTION V3	78.35%

CONCLUSION

Free-Form Visualization:

I took 6 images outside the training and testing set and tried on my model. These images have different resolution and size and the output is shown below:

All the predictions are matched with labels and we find that two breeds in almost every case could not be recognized correctly.

Reflection:

In the start I tried to see if I could build and train a CNN from scratch but then realized the dataset is small and even augmentation by rotating, etc would not help and then I discovered about transfer learning and the various pre- trained models and the ways in which these can be used.

I tried to implement three models and compare among the three. These three have different architectures and these models have their weights optimized for a wide degree of image classification problems. They are used as feature extractor by retraining the top layer adding a few more layers of our own and I found this best for my problem.

These keras frameworks takes away the hasty tasking of building everything from scratch and hence I could focus more on optimizing the model. I also realized that training and testing are better when done in batches. Since the basic model used trained on Image net, it can also recognize other objects too.

Improvement

There are ways in which this model can be improved further:

1) We have on an average 80 images per class, this data can be augmented or more images can be added.
2) We can use fine tuning of the models by retraining a few layers.