Capstone Project

Machine Learning Engineering - Nanodegree
Saurabh Agrawal
March 7, 2018

Dog Breed Identification

Definition

Project Overview

This project is based on a kaggle competition taken from https://www.kaggle.com/c/dog-breed-identification. The objective of the project is to train a model which can predict the breed of a dog based on the image. There are too many different breeds of dogs and it is not easy to identify which breed a dog belongs to. kaggle is providing a datasets of images which has strictly dog images. There are total 120 breeds of dog covered in this training set. Although with more dog breeds training data, this solution could be extended for more than 120 breeds as well. The training set has a total of 10222 images across these 120 breeds. The training and test data files for this project are available at https://www.kaggle.com/c/dog-breed-identification/data .

The purpose of this project is to train and test the model for 120 breeds of dogs only. I expect that this model would be easy to train on other breeds of dogs and

other animals like cats, horses, monkeys etc but this project does not aim to test this.

Problem Statement

There are more than 360 breeds of dogs as recognised by FCI (Fédération Cynologique Internationale) - https://en.wikipedia.org/wiki/List_of_dog_breeds_recognized_by_the_FCI.

This is one of those problems where if trained well, a model can easily outperform humans. A regular person can mostly identify the animal type (cat vs dog vs monkey) and differentiate between 4-5 popular breeds. Even for an expert, it might be difficult to differentiate between all the 360 breeds, let alone remember details of all those. This could be a common problem if one considers the breeds of other similar animals like cats, monkeys, horses, cows, pigs etc.

There is a certain level of added complexity that images have not only dogs but in some of the images there are other people also. Some of the images have dog with leash or a collar or an object in dog's mouth which might confuse the model.

Once trained, the model would be able to tell the breed of a dog in a new unseen image. The model is supposed to provide probability for each of 120 breed (class). The expectation is that model will be able to clearly assign a higher probability to the correct breed and lower probability to incorrect breed of dog.

Metrics

The output of the model for this problem will be a probability for each class. The probability of a class will be the chances of the dog belonging to that breed. As these are all mutually exclusive classes, i.e. a dog can belong to only one breed, the sum of all the probabilities should be 1.
For this problem, a multiclass log loss function will be ideal. Kaggle has also chosen the same for scoring the project. For test data Kaggle will not provide us the labels but the directly the log loss score.

For our model, the accuracy is the percentage of predictions where predicted value is equal to actual value. Accuracy considers prediction as a yes no feature and hence not a good metric for perfection of our model. On the other hand, log loss considers the probability of correct label. For multi classification problem, log loss tells us the confidence of our model. The target of our machine learning model would be to minimize the log loss value. A log loss value of 0 will mean a perfect model.

Log Loss function

Below is the curve of log loss vs accuracy in Fig1. For lower accuracy, log loss value is really high, but as predicted probability approaches 1 i.e. perfect outcome, log loss value approaches 0. -

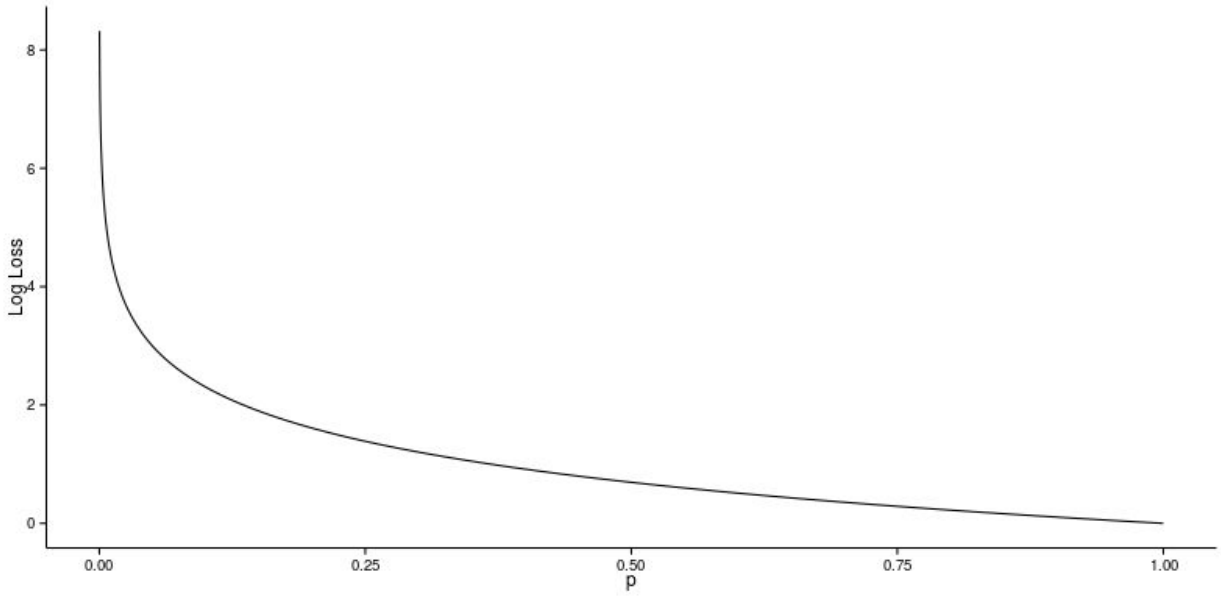


Fig 1 - Log Loss Vs predicted probability

Mathematically log loss function is defined as -

$$-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\ log\, p_{ij}$$

Where N is the number of instances, M is the number of classes, y_{ij} is binary indicator of whether or not label j is correct class for instance i and p_{ij} is the model's predicted probability that j is the correct class for instance i.

Analysis

Data Exploration

There are total 10222 images provided in training set. The training set images belongs to overall 120 classes (breed of dog). Average number of images per class is 85 while lowest number of images for a class is 66.

There is also a separate test set of images. There are total 10357 images in the test set. I do not have the labels for them as these set of images will be used by kaggle to calculate the score of the model.

Fig 2 has some images taken randomly from the training set to demonstrate the nature of images. Below are the observations from that -

- The resolution of the images is not same.
- Aspect ratio of images is also different for different images.
- Some images have other people along with dogs in the image (cell - 2x4, 3x2).
- Cell 2x1 has image of a dog with some wooden piece in mouth.
- Cell 3x3, 4x1, 4x3 have images with dogs wearing leash or some cloth.

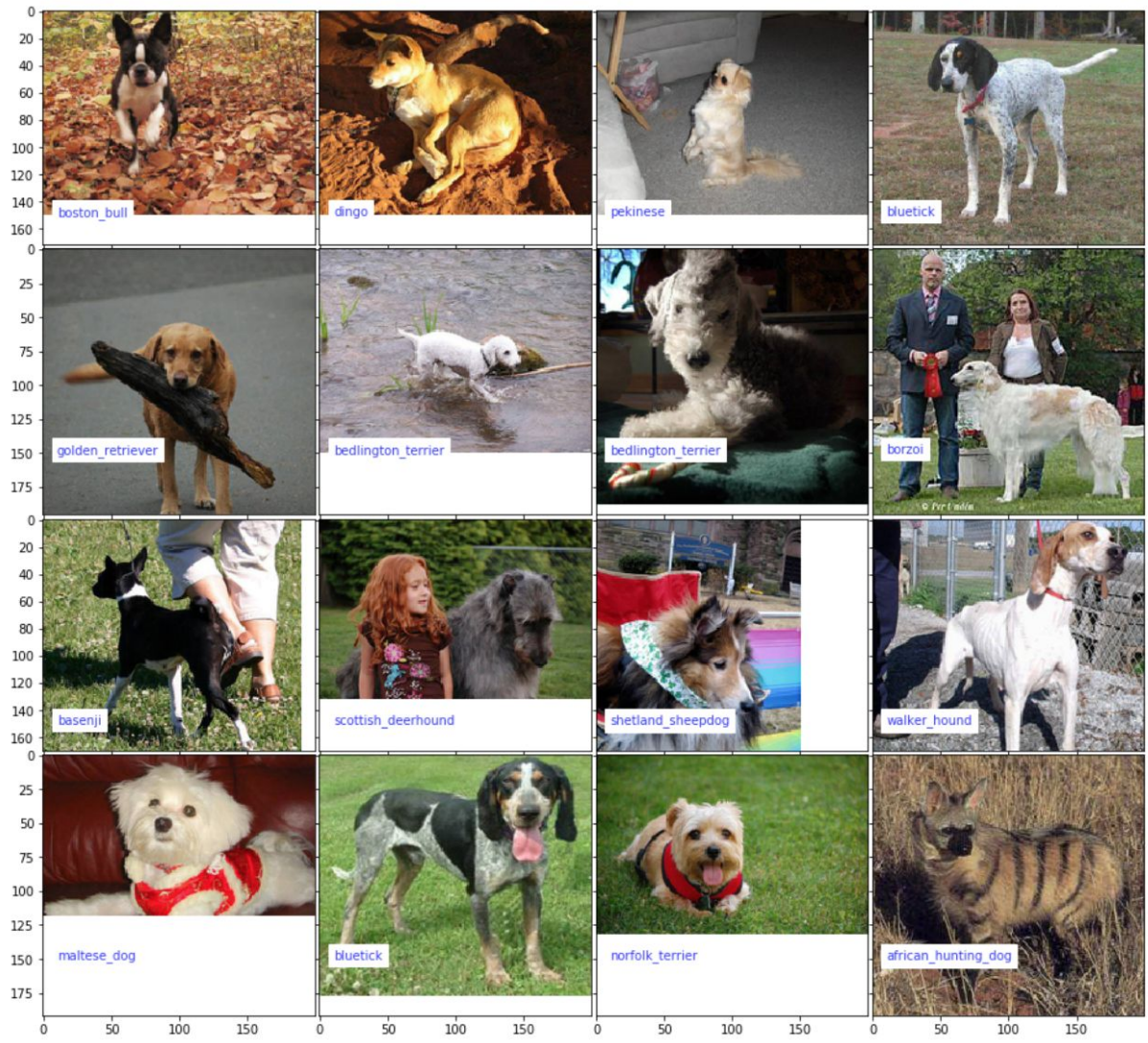


Fig 2 - random images from training set

Exploratory Visualization

Fig -3 provides a bar chart of number of images for each class. We have class names on x axis and number of images for that class on y-axis. Horizontal line at 85 shows the mean of these values.

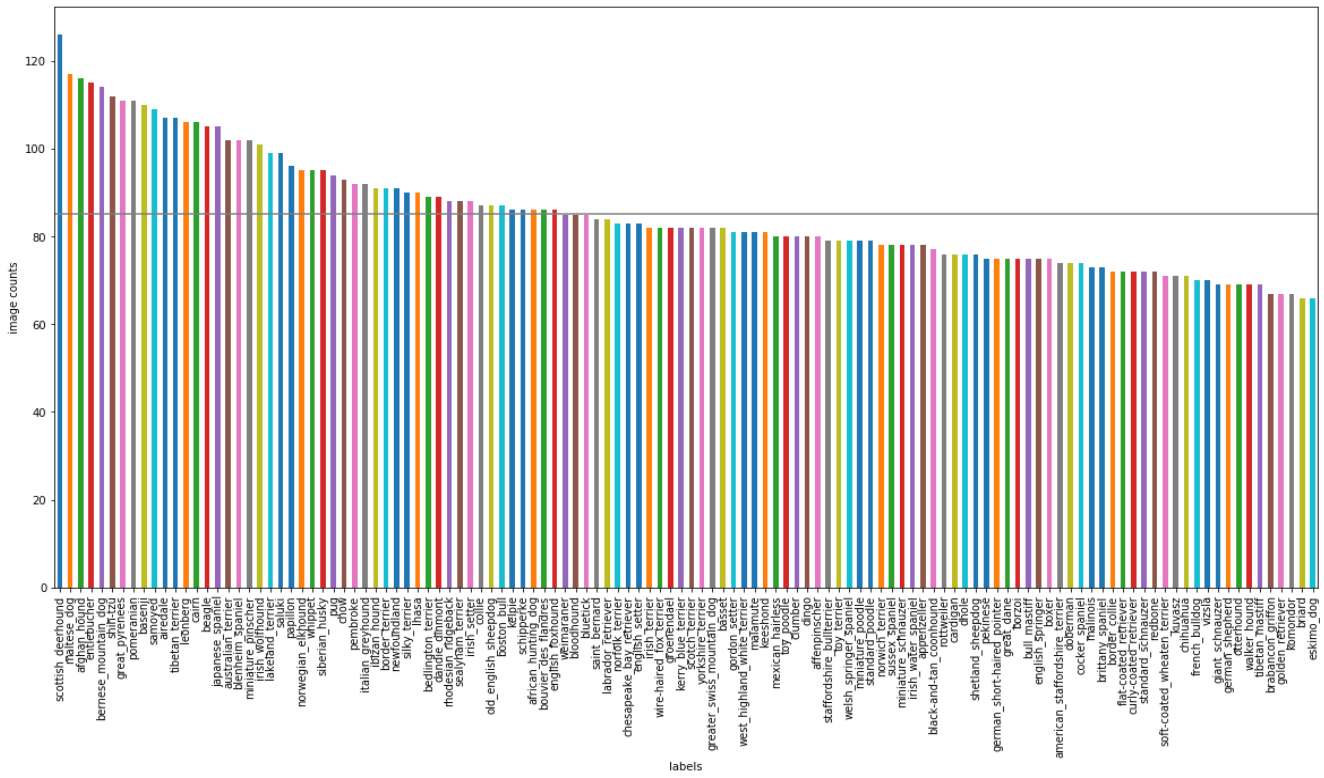


Fig 2- labels vs image counts

Dogs are not always in the center of image so we can not crop the images. It will be difficult to train a model with varying number of inputs. To avoid this, I will resize all the images to one common size before feeding to the model.

Algorithms

Convolutional Neural Net is a popular model for image processing and recognition problems. CNN are inspired from biological functioning of our visual cortex. Some parts of model detect horizontal edges, while some detect vertical edges. Deeper layers detect shapes and objects.

Technically speaking a CNN model takes the image as input, pass it through a series of convolutional, non linear, pooling and fully connected layers and then derive the output. Input to a CNN is usually a 3 dimensional pixel map a x b x 3 where first 2 dimension a and b are pixel location and third denotes the channel

(RGB). The output of this model could be a single class or a probability of the classes.

CNN Architecture

A CNN architecture defines a particular arrangement of these layers. A typical CNN architecture will look like -

Input -> conv -> relu -> conv -> relu ->pool -> relu -> conv -> relu -> pool -> fully connected -> softmax

Convolutional Layer (conv)

Convolutional layer is a filter applied on the input to this layer. Convolution layer filter will have the same depth as the input layer. A convolution layer can be imagined as someone scanning an image using a flashlight and sliding that flashlight slowly over the image. The conv filter multiplies the filter array with the pixel values in the region in focus and sum up the multiplication values. This process is repeated for the entire image as the filter slides over the image. Functionally each conv layer is like a feature identifier which tries to identify specific patterns in the input image.

Pooling layer (pool)

Pooling layer takes the feature map output of conv layer and prepares a condensed feature map. e.g. if we apply max pooling to 2x2 region, then it will output the maximum activation of 2x2 area. Another approach could be to use average pooling where it will output the average of 2x2 region.

Fully Connected Layer

Fully connected layer takes the input from a conv or pooling layer and outputs N dimensional vector where N is the number of classes. Fully connected layer basically takes the output of high level feature layers and predicts that probability of a class based on that feature.

Softmax Layer

For mutually exclusive classification problems, it helps to apply softmax layer as the final layer. It takes a vector as input and outputs another vector which has all the numbers in range 0 - 1 and they all sum up to 1. In effect, softmax function provides the probability of each class for a particular input.

There are many architecture popular for CNN. Some of the most popular architecture are ImageNet, AlexNet, Inception, Xception. Inception and Xception has had most success in image recognition tasks. For our classification problem here, I plan to choose Xception architecture as it has better success than inception.

Xception Architecture

Xception architecture is built on the hypotheses that mapping of cross channel correlation and spatial correlation can be decoupled. Xception architecture has 36 convolutional layers structured in to 14 modules. Overall Xception architecture can be divided into 3 stacks - entry flow, middle flow repeated 8 times, end flow. As we are training our model for a classification problem, there will be a logistic regression layer in the end.

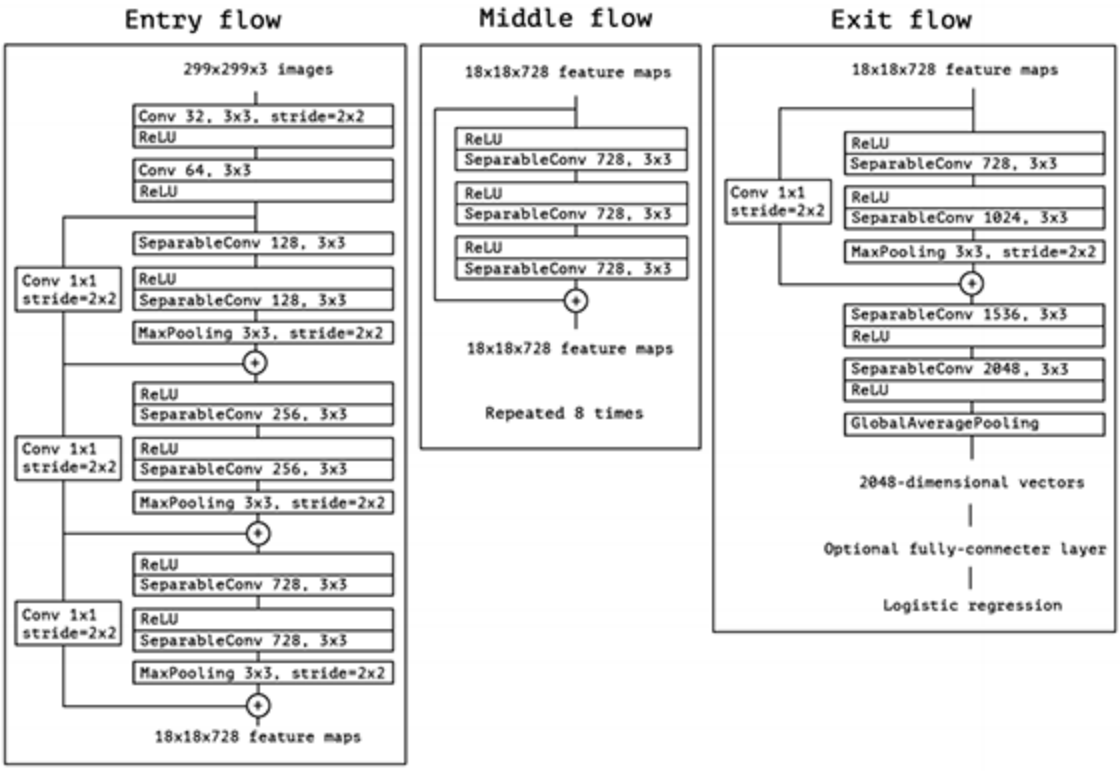


Fig 3 - Xception Architecture

Benchmark

The target is to achieve a loss of around 0.3 and to have the score better than at least 50% of kaggle leaderboard.

Methodology

Data Preprocessing

The resolution of images is not standardized. All the images have different images. Moreover standard input size for Xception is 299x299x3. All our images have resolution in format of AxBx3. So we need to resize the images before we start feeding it to our model. I am using preprocess_input method of keras.applications.xception class.

I also split data into 80:20 ratio using 80% of the data for training and 20% for validation. For testing there is a separate set of images provided by Kaggle. Testing set has 10357 images. I do not have the labels for the test set and for test set I upload the probability results to kaggle and get the score.

Implementation

Using Keras library, I instantiate a model of Xception without the top layer. This model would be used to get predictions for both training data and validation data. Then I trained another logistic regression model which was fit on the training data prediction and used validation data prediction for validating the logistic regression model predictions. Thus Logistic regression model acts as the top layer of Xception model.

I also have a DEBUG mode in my code. If DEBUG is set to True, then I take only top 16 classes and images belonging to these 16 classes for training and validation. Then I split this data into 80:20 for training and validation and try my model. DEBUG mode is helpful for testing the implementation and experimenting with the model. Due to reduced dataset, I am able to run DEBUG mode in very short time. Although accuracy is usually very high in debug mode because of reduced number of classes but it is really helpful for identifying the trend on log loss value and for tuning the model.

It would not be possible to load all the images and feed to CNN at the same time as system will run out of memory. To avoid this, I am loading only a smaller set of images, feeding it to CNN and then saving CNN prediction of it. The prediction object out of CNN is quite small. In the second step, I am able to feed entire prediction set out of CNN to my logistics regression model. In case, we have a very large number of images, then we can use Stochastic Gradient Descent in place of logistic regression for second step.

To get the final score from kaggle on the test set, I get the predictions on the test set using my model and store them in a separate csv file in the format defined by kaggle. Then I upload this file to kaggle to get the score on test set.

Refinements

For comparison, I chose a smaller dataset of 16 classes only so that I can run multiple iterations of refinement without spending and waiting too much time for process to finish.

I first tried with inception model and accuracy was 0.96 which is quite good. But when I tried Xception model, then accuracy was marginally higher around 0.97 and loss value was comparatively smaller.

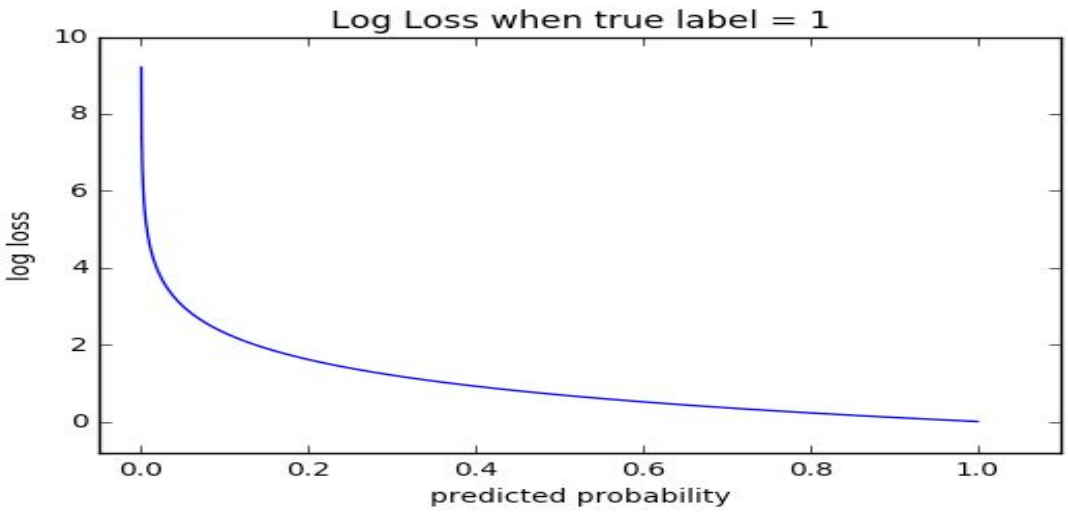
Another refinement I did was to use max pooling and avg pooling in Xception model. I tried both and log loss for max pooling was 0.4731 while that for avg pooling was 0.3348. Avg pooling had a considerably better results.

Results

Model Evaluation and Validation

My model was able to achieve a loss score of 0.3348 and accuracy of 89.9% . For test data, kaggle only provided the loss score which was 0.34205. Considering the loss score of my validation set and test set is very close, I can say that my model is trained well and performing well on unseen data as well. Test data is bigger than training data. Considering that model is performing well on both train and test data, we can conclude that model is trained well.

Below image provides the log loss vs predicted probability graph. A log loss score of 0.34 means a predicted probability of around 0.8 which is good for our algorithm.



Justification

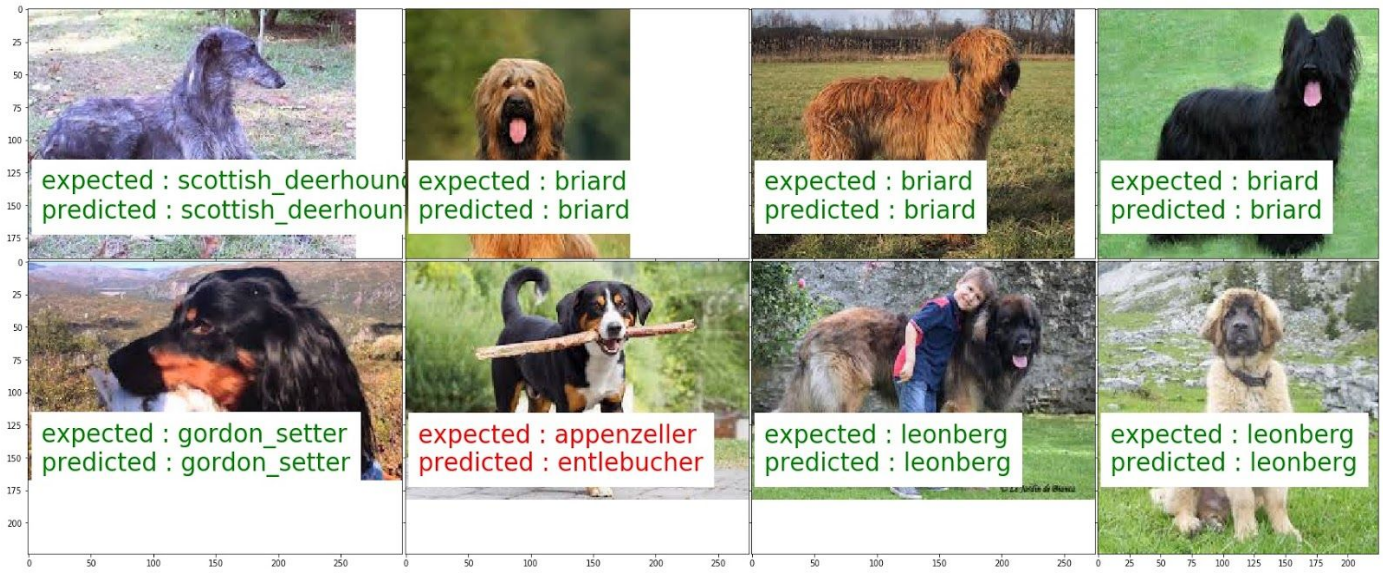
The log loss value of our model is 0.3348 which is very close to the target value we set earlier (0.30) . The accuracy of our model is 89.9%. The accuracy of Xception model is 79% although on a much higher number of classes. On kaggle leaderboard, this score is within top 550 among more than 1200 entries. I believe this could have been even higher if kaggle had a private dashboard where it would have been evaluated on an unknown test set.

Considering a smaller data set for training, this is a good score. The solution is good and can be further improved by applying Data augmentation and other advanced methods.

Conclusion

Free-Form Visualization

I took another 8 images outside the training or test set and tried on my model. These 8 images are consistent with training dataset in the sense that they all have images of dog and the dog breed is one of the classes in our problem. Resolution of images is also not consistent and all the images are of different sizes.



In the above grid, correct predictions are in green and wrong predictions are in red. Except one image, all other images have correct prediction. The model seems to be working well even for images outside our dataset.

If we can improve accuracy of our model little bit more using data augmentation or more training data, this model could very well be used for practical applications.

Reflection

In the start I tried to build and train my own CNN model from scratch but realized that training data set is too small to train it. While reading for best suitable architecture for image classification, I came to know about inception and Xception architecture and pretrained models in these architecture. These pretrained models already have their weight optimized for a wide degree of image classification problems. They can not be used as it is but I could retrain their top layer and use them for my problem. This was perfect for my problem.

Keras library take away lot of boilerplate code and make it easy to use these pretrained models. This allowed me to focus more on optimizing the model.

While training the model, I realized that images can hog the memory really fast. To keep memory consumption low, I fed the training and test data to my model in small batches. This made the execution really fast. I think there is still lot of room to make my execution faster. Training the model took lot of time and in hindsight I could have spend some more time to make the execution faster and thus save time in repeated training of model for optimization.

It is amazing that I was able to train for 120 classes with so few images. This demonstrates how well pretrained model works. With advance optimization methods this could be even better.

Although I trained this model on dog breed classification only, but this same model could be trained for any similar problem of image classification of other animals or even vehicles, planes etc.

Improvements

There could be ways to improve this model further. The training dataset for training this model was really small. We had on average 85 images per class. If we could get more images to train the system, then its accuracy should definitely improve. Another approach could be to use data augmentation where new images will be generated from the existing set of images itself by randomly flipping them at different angles or by cropping the existing images.

References

https://www.tensorflow.org/tutorials/image_recognition
https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/
https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/
http://openaccess.thecvf.com/content_cvpr_2017/papers/Chollet_Xception_Deep_Learning_CVPR_2017_paper.pdf
http://neuralnetworksanddeeplearning.com/chap6.html#introducing_convolutional_networks