**Dog Breed Identification using Deep Learning**

***Bring a dog breed classifier model when you go to the dog park.***

**Introduction**

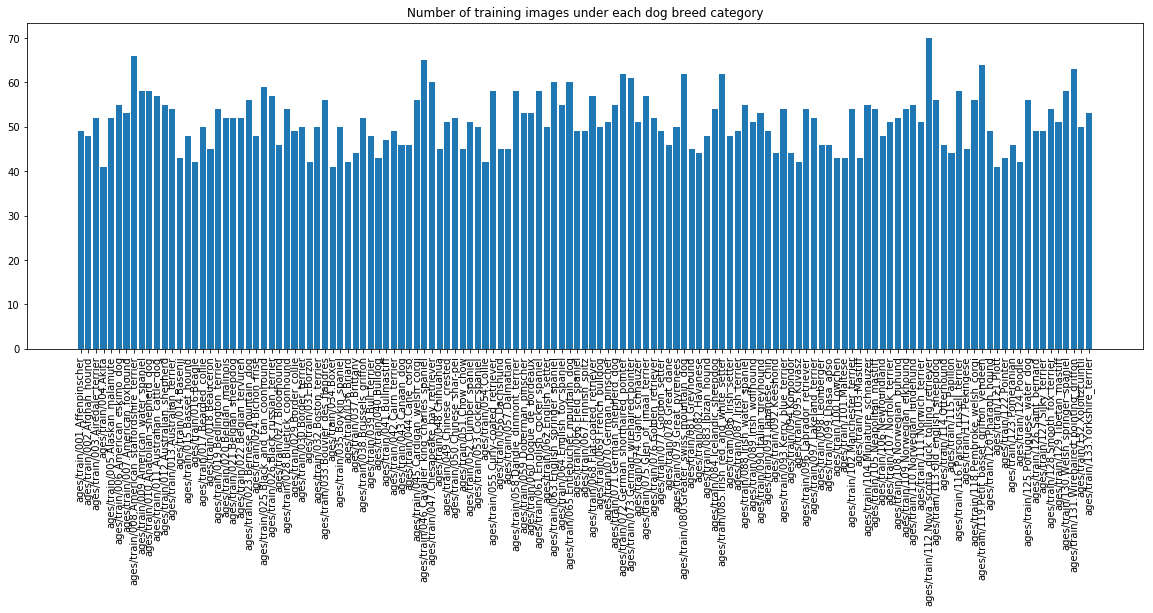
The goal of this project is to detect the dog’s breed using an image. For the fun part, the system will first detect if the image is of a dog or a human’s face. In case of a dog, it will predict the dog’s breed. In case of a human, it will predict the dog breed which the human most resembles. This project was developed as a capstone project for Data Scientist Nanodegree at Udacity.

Deep Learning technology has out-smarted every other conventional Artificial Intelligence algorithm in solving problems that involve perception. As humans, we give credit to evolution that enabled us to see, listen, smell and feel the world around us. Every day we see several examples of Deep Learning technology enabling computers to see, listen, and speak! These narrow AI solutions may not be as intuitive as humans, but it is fascinating to witness how these mathematical models, with a small amount of time and effort, can perform magic!

The choice of using Convolutional Neural Networks and OpenCV is obvious in this case. The model was trained on the dataset provided by Udacity, that contains 133 dog breed categories and 8351 images. And I utilized the GPU-enabled Udacity workspace with Keras and Tensorflow to train the models.

**Data Exploration & Visualizations**

The Humans dataset, also provided by Udacity, contained 13233 images. The dog dataset has 133 distinct dog breed categories and 8351 total dog images. Among these, 6680 images were used for training, 835 for validation, and 836 for testing.



As you can see, the dataset is balanced with an average of 50 images per dog breed class. Since this is a classification problem, I chose to use accuracy as an evaluation metric. If the dataset was overly imbalanced, then precision would have been an appropriate choice.

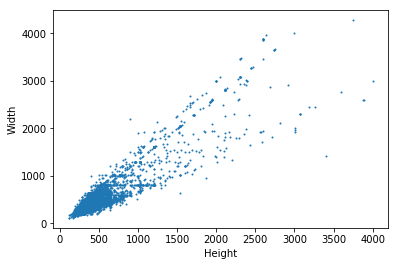
Let’s look at these dog images.







The individual images were of different sizes and orientations, but most of them were in the range of 300-500 pixels in height and width. As a part of pre-processing step these images were resized to 224x224 pixels to fit the network architecture. The distribution of height and width is shown below.



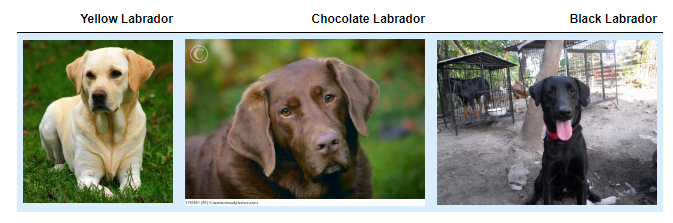
**Is it a Dog or a Human?**

OpenCV’s Haar feature-based cascade classifiers was used to detect if the image is a human face or not. This detector was tested on the first 100 images from the human and the dog datasets. It achieved 100% accuracy on the human images, but it also detected 11% of the dog images as human faces.

To detect dogs, I used a pre-trained model [ResNet-50](http://ethereon.github.io/netscope/#/gist/db945b393d40bfa26006) model that was trained on the famous ImageNet dataset. The ResNet-50 model has the same 133 dog breed categories as part of its output classes. The dog detector was tested on the same set of images mentioned above and it achieved 100% accuracy on the dog images, and it didn’t detect any dogs in the human face dataset.

I want to mention that detecting the dog’s breed is a challenging task. Even a human would have difficulty in distinguishing between a Brittany and a Welsh Springer Spaniel. Also, labradors come in three shades - yellow, chocolate, and black. Our vision-based algorithm will have to conquer this high intra-class variation to determine how to classify all these different shades as the same breed. Well if you are a dog expert, then you may have some advantage.





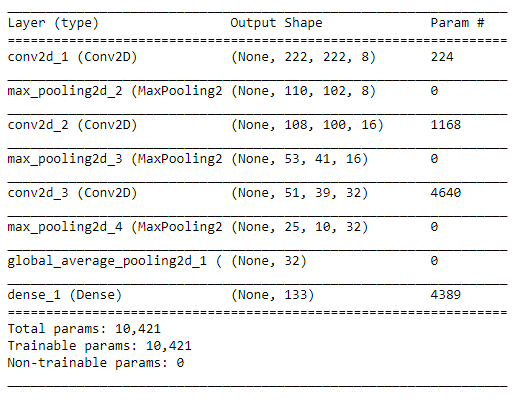
**Data Pre-Processing**

When using TensorFlow as backend, Keras CNNs require a 4D array as input, with shape ***(nb\_samples, rows, columns, channels)*** where ***nb\_samples*** corresponds to the total number of images, and ***rows***, ***columns***, and ***channels*** correspond to the number of rows, columns, and channels for each image, respectively. As we are dealing with color images, the number of channels is 3. And we also resize all the images to a square image of size ***224×224*** pixels.

A pre-processing function was implemented to achieve this task. The output of this function is a 4D array of shape ***(nb\_samples, 224, 224, 3)***. As I plan to use pre-trained models, this function also included an additional normalization step that subtracts the mean pixel (expressed in RGB as [103.939,116.779,123.68] from every image.

**Create a CNN from Scratch**

The first step is to attempt a CNN model from scratch. Given the complexity of the problem, training a CNN model that achieves good accuracy will require a lot of time and computing resources. Given the intention of our project, lets define the limits. As we have 133 output classes, a random choice is below 1% accuracy. So, anything better than random is a success.



I used the above CNN architecture for the first experiment. I defined a few convolutional filters of increasing size (8, 16, 32), which is a standard practice in building CNNs.

We can take a safe assumption that the neighbouring pixels are similar in our images and use MaxPooling layer to reduce the dimensions of input images along CNN layers. This will also reduce the chances of overfitting but will not affect our model's prediction capability to a great extent. I chose stride length of 2 as it ran much faster.

And I also used GlobalAveragePooling as it is recommended in CNNs to reduce spatial dimensions.

I trained the model for only 5 epochs as it achieved a test accuracy of 4.3%. We can continue to improve our model by adding more layers and training for a greater number of epochs. But we can also be smart and utilize transfer learning to its advantage.

**Transfer Learning to the rescue**

***If I have seen further than others, it is by standing upon the shoulders of giants.***

* ***Sir Isaac Newton***

Transfer learning allows us to utilize the knowledge gained from solving one problem to solve a different but related problem. For example, we can use the models trained on ImageNet data to solve the dog breed classification problem. Here, we retain the layers correspond to feature extraction part from the pre-trained models and train only the final classification layers. Since feature extraction is the most excruciating part of model training, reusing it allows us to achieve good results with less computational resources and time.

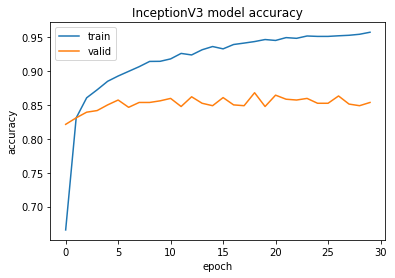
First, I used the bottleneck features from VGG16 model and added just the global average pooling and dense layers at the last. I was able to achieve training accuracy of 43% and test accuracy of 35% after training only for 20 epochs and each epoch took only a couple of seconds. I also observed that the validation accuracy kept increasing as I trained for more epochs.

Then I used bottleneck features from InceptionV3 model to train another classifier. Like my previous experiment, I added a GlobalAveragePooling layer which retains only the important information and reduces the dimensionality to a great extent. The final dense layer is used to predict the 133 dog breed categories.

The InceptionV3 is trained on ImageNet dataset consisting of 15 million images, but we have roughly 6000 images in our dataset. This difference in dataset sizes would make the model to overfit, so I added a Dropout layer to reduce overfitting.

**Evaluating the model**

The training and validation accuracies for first 30 epochs is shown below.



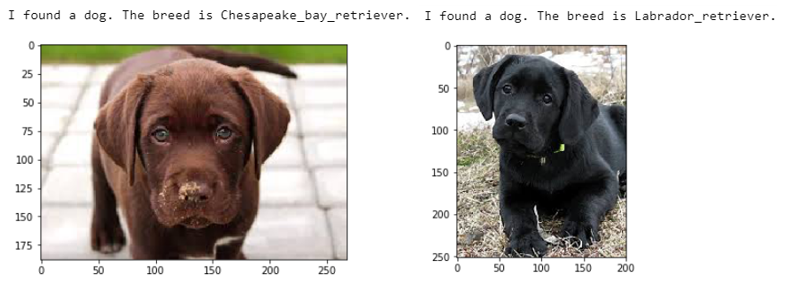
From this graph, we can see that the model is still overfitting. This may be due to the relatively small size of our training dataset. We have 133 categories and only 6680 training images which means only a small set of images for each category.

The model achieved the test accuracy of 78% after training for 30 epochs. Training for more epochs or tweaking model architecture may not improve our results further. The smarter approach is to experiment with data augmentation to increase variance of our training data which would improve validation accuracy.

When tested on new images, my model performed as I expected, not perfect but good enough. As we see from these test images, my model can detect dog breeds correctly with high inter-class variation. i.e. It detected the breeds Brittany and Labrador accurately.



It fails when it comes to dog breeds with low inter-class variation. i.e. It detected a Chocolate lab as 'Chesapeake\_bay\_retriever' and a Black lab as 'Labrador\_retriever', but I'm glad that it was able to identify the breed as the retriever.



Human face test was the fun part and the results were close enough.

Next step would be to perform data augmentation to increase variations in training dataset and improve the model accuracy.

**Conclusion**

We started with a scratch CNN model with accuracy of 3% and trained a model with 78% accuracy using transfer learning and little effort is impressive. The accuracy may be seeming low, but our problem is not very sensitive to be bothered about that. Given time and effort, we can achieve above 90% accuracy in this problem using the transfer learning and data augmentation methods.

GitHub Repository can be found at:

<https://github.com/RekhaChandrasekaran/DSND_Capstone_Dog_Breed_Classifier>

**References**

1. Narrow AI

<https://en.wikipedia.org/wiki/Weak_AI>

1. Haar feature-based cascade classifiers <http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html>
2. Intro to Global Average Pooling

<https://adventuresinmachinelearning.com/global-average-pooling-convolutional-neural-networks/>

1. VGG16

<https://neurohive.io/en/popular-networks/vgg16/>

1. InceptionV3

<https://medium.com/@sh.tsang/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015-17915421f77c>