Machine Learning Engineer Nanodegree

Capstone Project

Name: Ahmed Monjurul Hasan

Date: Mar 12, 2020

Project Overview

There are thousands of different dog breeds in the world. Some of these dog breeds are too close to differentiate from their images. For example, the above two dogs seem to belong to the same breed. But, the left one is Norfolk Terrier, and the right one Norwich Terrier. With the advancement of technology, computers are getting intelligent enough to distinguish different classes of objects from their images. Since 2010, the ImageNet project runs an annual software contest, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where software programs compete to correctly classify and detect objects and scenes using one thousand non-overlapping classes [1]. In this capstone project, I want to leverage the state of art of image classification model on Imagenet to teach computers to predict 133 dog's breed from an image using deep learning algorithms.

Problem Statement

I want to go above and beyond the dog's breed classification problem by tackling two minor but interesting problems as well such as human face detection and dog detection using computer vision and machine learning techniques. Using these three solutions, I would like to build an application that could be used as part of a mobile or web app. This application will accept any user-supplied image as input. If a dog is detected in the image, it will provide an estimate of the dog's breed. If a human is detected, it will provide an estimate of the dog breed that is most resembling. If it detects neither a dog nor a human, it will give an appropriate error message.

Metrics

Since we are dealing with a multi-classification problem here and the data is slightly imbalanced, I used accuracy evaluation metric and negative log-likelihood loss function. The main objective in a learning model is to reduce (minimize) the loss function's value with respect to the model's parameters by changing the weight vector values through different optimization methods, such as backpropagation in neural networks.

Loss value implies how well or poorly a certain model behaves after each iteration of optimization. Ideally, one would expect the reduction of loss after each, or several, iteration(s).

The accuracy of a model is usually determined after the model parameters are learned and fixed and no learning is taking place. Then the test samples are fed to the model and the number of mistakes (zero-one loss) the model makes are recorded, after comparison to the true targets. Then the percentage of misclassification is calculated.

For example, if the number of test samples is 1000 and model classifies 950 of those correctly, then the model's accuracy is 95.0%.
[Ref](https://stackoverflow.com/questions/34518656/how-to-interpret-loss-and-accuracy-for-a-machine-learning-model)

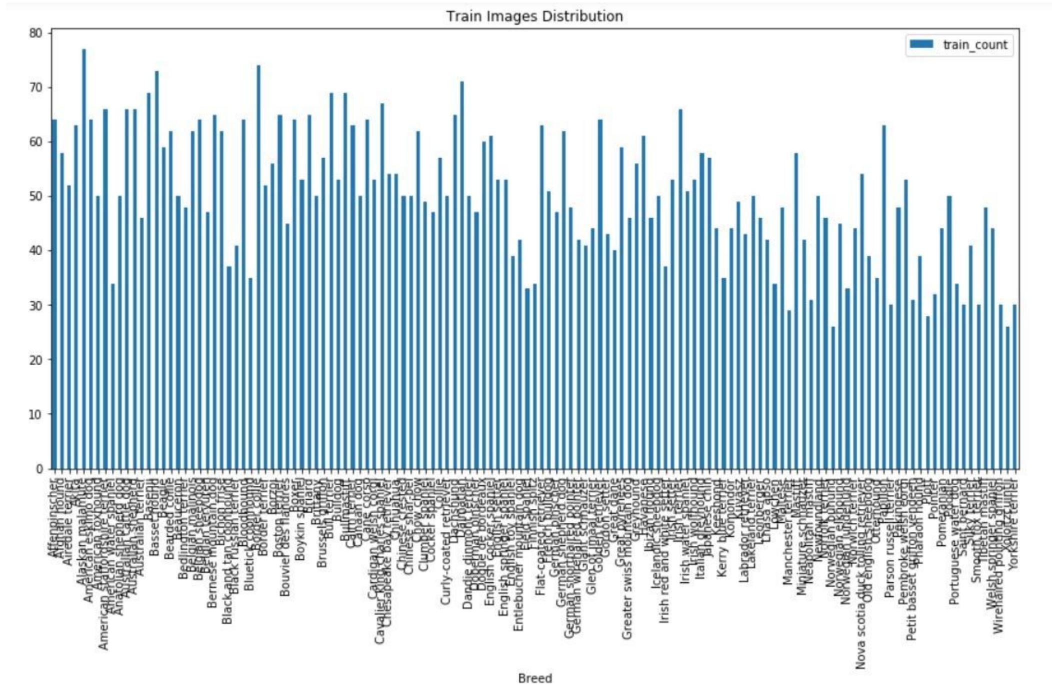
Also, we have employed 3 types of error metrics namely top-1, top-3 and top-5 error. In the case of top-1 error, we check if the top class (the one having the highest probability) is not the same as the target label.

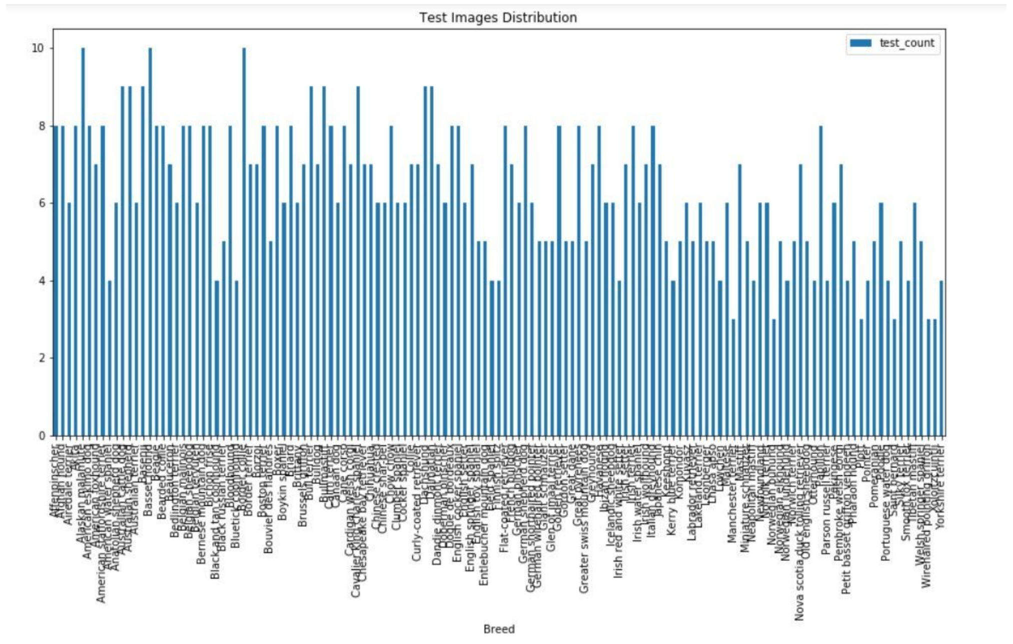
In the case of top-3 errore, we check if the target label is not one of our top 3 predictions (the 3 ones with the highest probabilities)

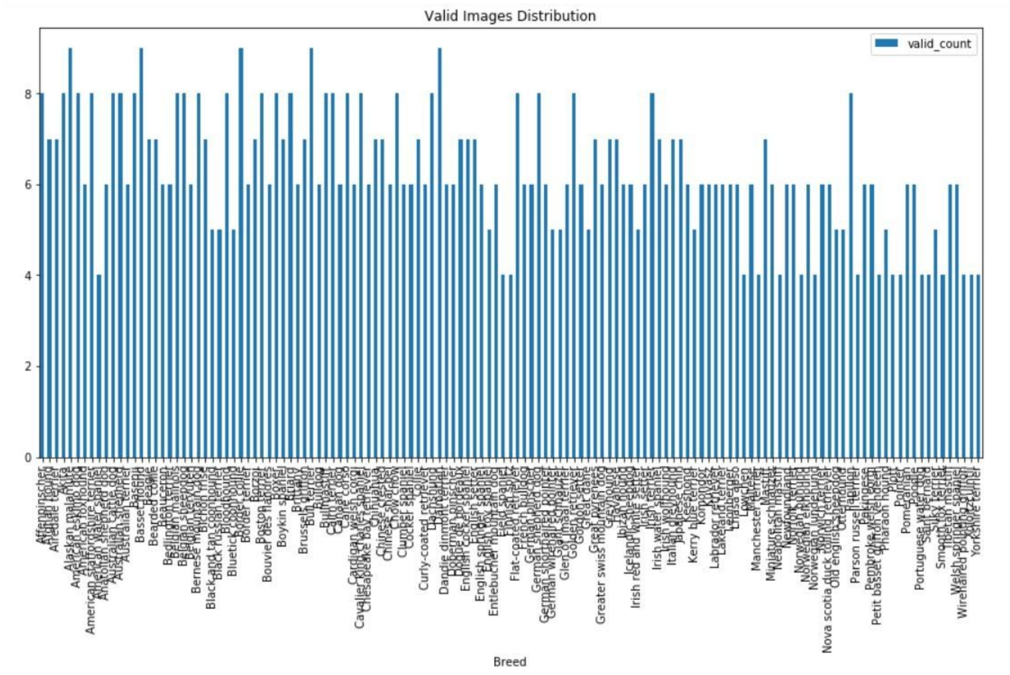
In the case of top-5 error, we check if the target label is not one of our top 5 predictions (the 5 ones with the highest probabilities).

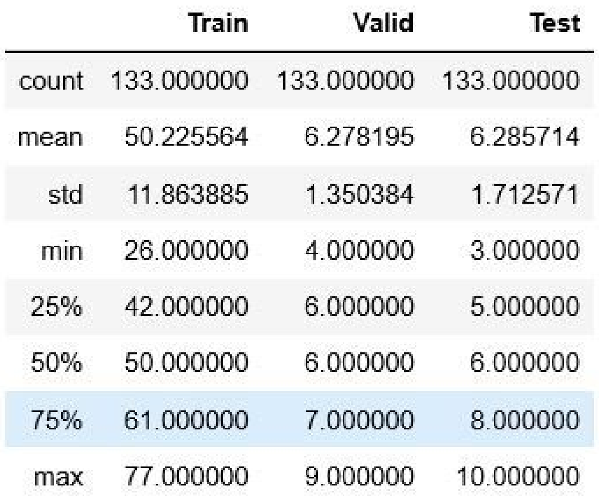
Data Exploration and Visualization

The dog dataset has 6680 images for training, 835 images for validation and 836 images for testing of 133 breeds. All these images have RGB channels and different sizes. So, they need to normalized and resiaed before using in the deep learning model.
From the train, test and valid dataset of dog images, we get the following distribution and statistics from ‘dog_app2.ipynb’:









	Train	Valid	Test
count	133.000000	133.000000	133.000000
mean	50.225564	6.278195	6.285714
std	11.863885	1.350384	1.712571
min	26.000000	4.000000	3.000000
25%	42.000000	6.000000	5.000000
50%	50.000000	6.000000	6.000000
75%	61.000000	7.000000	8.000000
max	77.000000	9.000000	10.000000

Algorithm and Techniques

The solution of the project is divided into following steps:
In the first step, I'll use OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images. In the second step, I'll use a pre-trained VGG16 model to detect dogs in images. In the third step, I'll create a CNN that classifies dog breeds. I'll create my CNN from scratch to attain a test accuracy of at least 1%. In the fourth step, I'll use the pre-trained VGG-16 model as a fixed feature extractor, where the last convolutional output of VGG-16 is fed

as input to my model. In Step 5, I'll use transfer learning to create a CNN using VGG-19 bottleneck features. In the last step, I'll write an algorithm that accepts a file path to an image and first determines whether the image contains a human, dog, or neither. Then, if a dog is detected in the image, return the predicted breed. If a human is detected in the image, return the resembling dog breed. Otherwise, it provides output that indicates an error.

Benchmark Model

Our models were compared with different benchmark data in a kaggle competition(https://www.kaggle.com/c/dog-breed-identification/discussion).The first model used VGG-16 bottleneck features and the second model used VGG-19 bottleneck features. Out of 4545 test images from the benchmark data, the first model misclassified 1193 images as top-1 error, 425 images as top-3 error and 254 images as top-5 error which gives a top-1 error rate of 26.25%, top-3 error rate of 9.35% and top-5 error rate of 5.85%. On the contrary, the second model misclassified 862 images as top-1 error, 247 images as top-3 error and 99 images as top-1 error which gives a top-1 error rate of 18.97%, top-3 error rate of 5.43% and top-5 error rate of 2.18%.

Model	Accuracy	Top-1 Error	Top-3 Error	Top-5 Error
Pre-trained VGG16	73.75%	26.25%	9.35%	5.85%
Pre-trained VGG19	81.03%	18.97%	5.43%	2.18%

Data Preprocessing

Each image in the train, valid and text dataset goes thorugh a number of preprocessing steps:

1. Train and Valid Dataset images goes through a series of transforms.RandomRotation(10), transforms.Resize(256), transforms.CenterCrop(224),transforms.RandomHorizontalFlip()
1. Test Dataset images goes through a series of transforms.Resize(256) and transforms.CenterCrop(224)
1. Normalize: Each image is normalized by mean=[0.485, 0.456, 0.406] and std=[0.229, 0.224, 0.225]

Implementation

The implementation of the project is divided into following steps:

Step 1: Detect Humans

We use OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images. OpenCV provides many pre-trained face detectors, stored as XML files on github. We have downloaded one of these detectors and stored it in the haarcascades directory.

Before using any of the face detectors, it is standard procedure to convert the images to grayscale. The detectMultiScale function executes the classifier stored in face_cascade and takes the grayscale image as a parameter.

Step 2: Detect Dogs

We use a pre-trained VGG16 model to detect dogs in images. Our first line of code downloads the ResNet-50 model, along with weights that have been trained on ImageNet, a very large, very popular dataset used for image classification and other vision tasks. ImageNet contains over 10 million URLs, each linking to an image containing an object from one of 1000 categories. Given an image, this pre-trained VGG16 model returns a prediction (derived from the available categories in ImageNet) for the object that is contained in the image.<br/>
While looking at the [dictionary](https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a), you will notice that the categories corresponding to dogs appear in an uninterrupted sequence and correspond to dictionary keys 151-268, inclusive, to include all categories from `'Chihuahua'` to `'Mexican hairless'`. Thus, in order to check to see if an image is predicted to contain a dog by the pre-trained VGG16 model, we need only check if the `ResNet50_predict_labels` function above returns a value between 151 and 268 (inclusive).

Step 3: Create a CNN to Classify Dog Breeds (from Scratch)

Now that we have functions for detecting humans and dogs in images, we need a way to predict breed from images. In this step, we create a CNN that classifies dog breeds. We create our CNN from scratch to attain a test accuracy of at least 1%.

Step 4: Use a CNN to Classify Dog Breeds (using Transfer Learning)

To reduce training time without sacrifysing accuracy, we have used the pre-trained VGG-16 model as a fixed feature extractor, where the last convolutional output of VGG-16 is fed as input to our model. We only add a global average pooling layer and a fully connected layer, where the latter contains one node for each dog category and is equipped with a softmax.

Step 5: Create a CNN to Classify Dog Breeds (using a different Transfer Learning)- Refinement

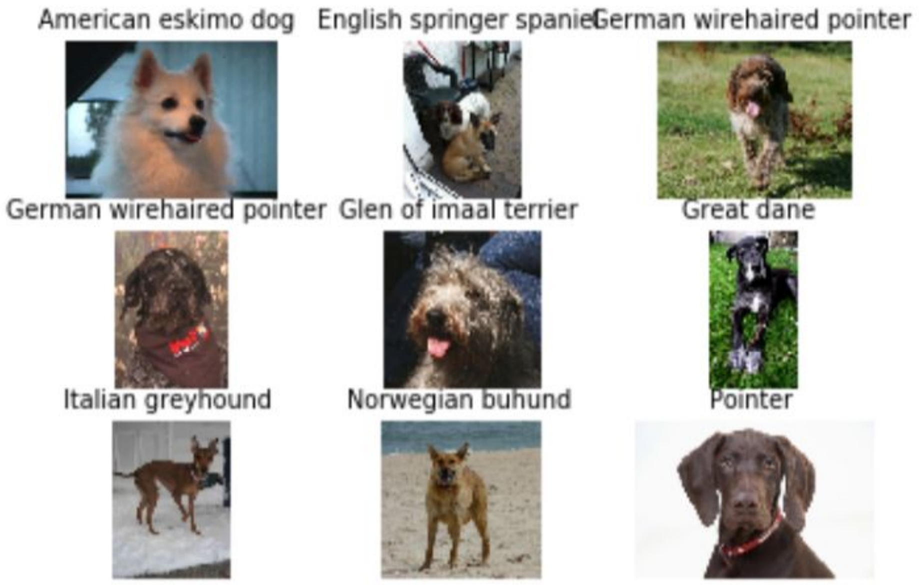
In Step 5, we used transfer learning to create a CNN using VGG-19 bottleneck features. In this section, we used the bottleneck features from a different pre-trained model such as VGG19.

Step 6: Write Algorithm

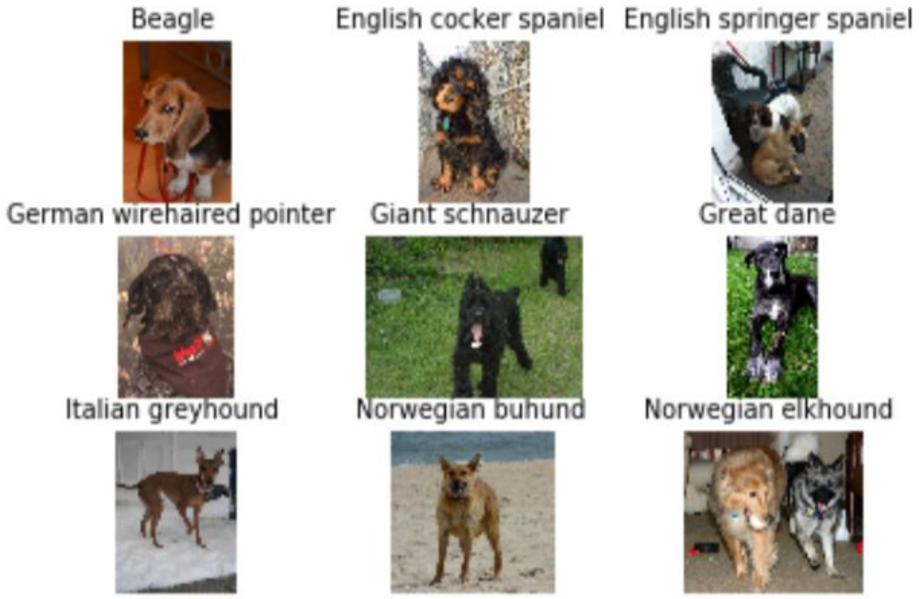
In this step, we write an algorithm that accepts a file path to an image and first determines whether the image contains a human, dog, or neither. Then,
* if a dog is detected in the image, return the predicted breed.
* if a human is detected in the image, return the resembling dog breed.
* if neither is detected in the image, provide output that indicates an error.

Refinement

In our first design, we have used transfer learning to create a CNN using VGG-16 bottleneck features. As a refinement, in our second model we have used transfer learning to create a CNN using VGG-19 bottleneck features. Second model gives slightly better results with better performance metrics. Please refer to the [next section](#eval) for a detailed comparison. Here are some misclassified images (top-5 error images) from the first model prediction:



Here are some misclassified images (top-5 error images) from the second model prediction:



From these two figures, we can see that out of these 9 images, 5 images were misclassified by both models. But each model made some unique mistakes as well from the other model.

Model Evaluation and Validation

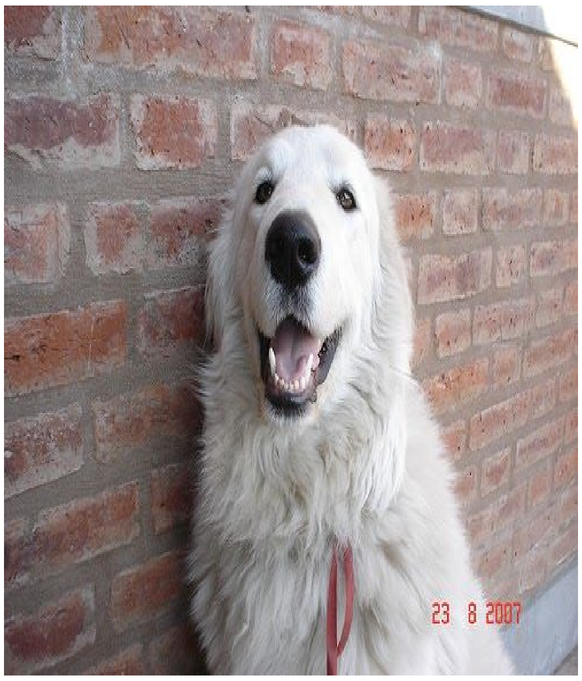
Now, we are going to show the comparison of performance metrics of two of our models using the transfer learning techniques. The first model used VGG-16 bottleneck features and the second model used VGG-19 bottleneck features. Out of 836 test images, the first model misclassified 128 images as top-1 error, 21 images as top-3 error and 10 images as top-1 error which gives a top-1 error rate of 15.3%, top-3 error rate of 2.5% and top-5 error rate of 1.19%. On the contrary, the second model misclassified 110 images as top-1 error, 17 images as top-3 error and 9 images as top-1 error which gives a top-1 error rate of 13.5%, top-3 error rate of 2.05% and top-5 error rate of 1.07%.

Model	Accuracy	Top-1 Error	Top-3 Error	Top-5 Error
Pre-trained VGG16	84.70%	15.30%	2.50%	1.19%
Pre-trained VGG19	86.85%	13.15%	2.03%	1.07%

The performance can be further improved by using some other pretrained model such as Densenet, Resnet or Inception models.

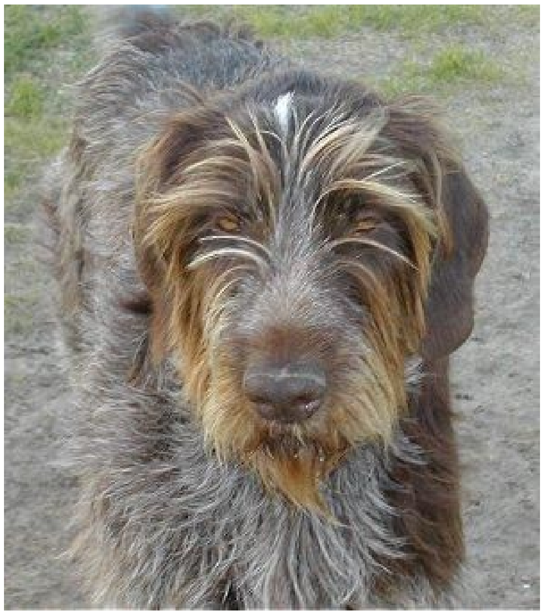
Justification

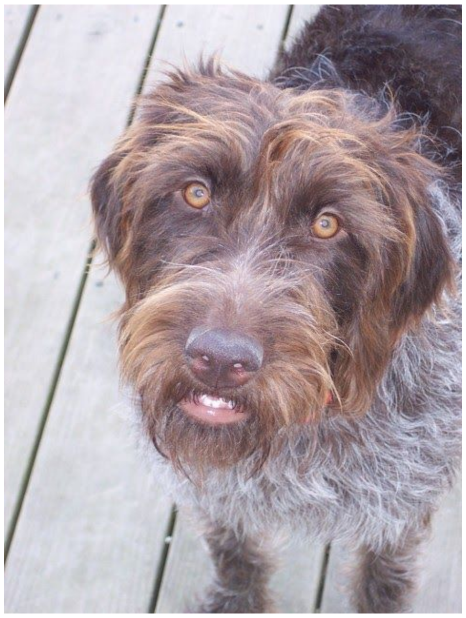
Due to less number of dog images of certain breeds, the model finds it difficult to predict some breeds.
We have observed that the model couldn't classify between Great pyrenees and Kuvasz, which both are white, big, and fluffy.



[Great pyrenees] [Kuvasz]

Also we have found the model fails to correctly classify german wirehaired pointer and wirehaired pointing griffon which look quite similar.





[german wirehaired pointer] [wirehaired pointing griffon]

It also couldn't distinguish between Mastiff and Bullmastiff, which is a mix between a bulldog and a mastiff.

[Bulldog] |[Mastiff] [Bullmastiff]

Reflection

These are the intersting or difficult aspects of this present application:

1. GPU TRAINING: The training requires a lot of computational power and hence it is impossible to do the project without GPU. I have done the training on my local laptop equipped with GTX 1060 GPU. Even though it has 6GB RAM, but it happened that when I tried to train two transfer models on the same notebook, the GPU ran out of memory. So we train different transfer models in different notebooks to overcome the situation.

2. WEB DEPLOYMENT: Due to some credit problem, we haven't tried to deploy our application in an external server. But we have used ngrok, which provides us a public URL for our application running in a local web server.

3. KERAS vs PYTORCH: Due to my previous familiarity with Pytorch, I have implemented all models using Pytorch instead of Keras with the approval of my mentor.

Improvement

These are the improvement ideas on this present application:

1. AUGMENT THE TRAINING DATA: Augmenting the training and/or validation set might help improve model performance.

2. DIFFERENT PRETRAINED MODEL: In our future project, we would like to use other pretrained models such as Densenet, Resnet or Inception.

3. OVERLAY DOG EARS ON DETECTED HUMAN HEADS: Overlay a Snapchat-like filter with dog ears on detected human heads. You can determine where to place the ears through the use of the OpenCV face detector, which returns a bounding box for the face. If you would also like to overlay a dog nose filter, some nice tutorials for facial keypoints detection exist [here](https://www.kaggle.com/c/facial-keypoints-detection/details/deep-learning-tutorial).

4. ADD FUNCTIONALITY FOR DOG MUTTS: Currently, if a dog appears 51% German Shephard and 49% poodle, only the German Shephard breed is returned. The algorithm is currently guaranteed to fail for every mixed breed dog. Of course, if a dog is predicted as 99.5% Labrador, it is still worthwhile to round this to 100% and return a single breed; so, you will have to find a nice balance.

5. AWS, GCP or AZURE DEPLOYMENT: It would be a good learning experience if we could depoly our web application to AWS, GCP or AZURE platform.

Reference

[1] Olga Russakovsky*, Jia Deng*, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei. (* = equal contribution) ImageNet Large Scale Visual Recognition Challenge. IJCV, 2015.