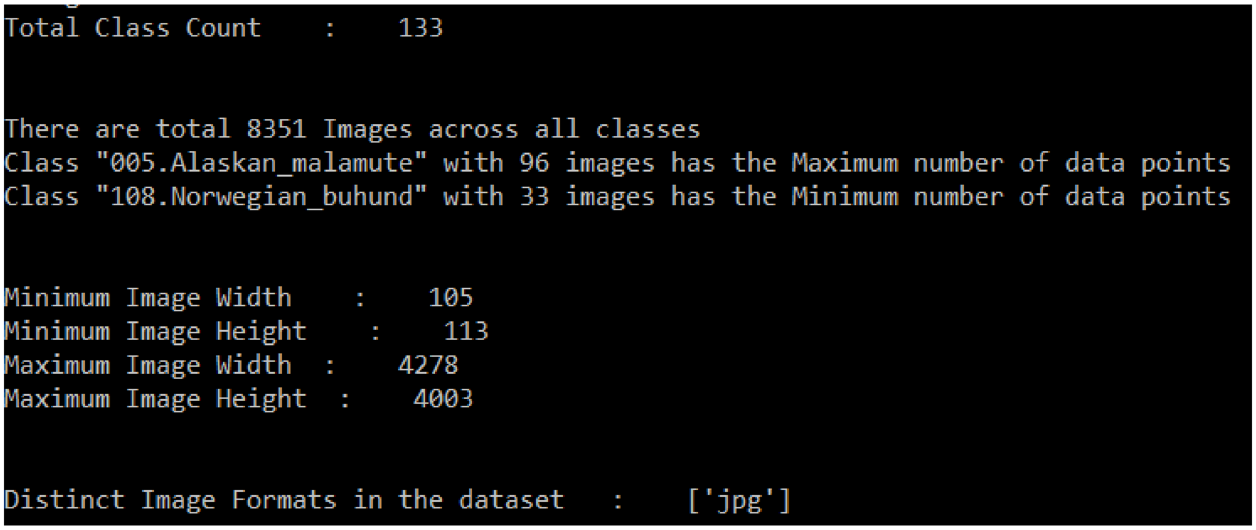
Dog Breed Classifier

The dataset:

The dataset consists of multiple images of different Dog Breeds. The images are organized into multiple folders, each folder representing a class (a specific Dog Breed) and all the images in the folder are of the dog breed the class represents.

I decided to run some simple stats on the dataset just to get a hang of it.



```
Total Class Count    :    133


There are total 8351 Images across all classes
Class "005.Alaskan_malamute" with 96 images has the Maximum number of data points
Class "108.Norwegian_buhund" with 33 images has the Minimum number of data points


Minimum Image Width    :    105
Minimum Image Height    :    113
Maximum Image Width  :    4278
Maximum Image Height  :    4003


Distinct Image Formats in the dataset   :    ['jpg']
```

(ref :: File: Analyze_Data.py :: Method: get_raw_data_stats())

My observations:

1. There are a total of 133 classes and only 8351 images in total. There are too many classes and too few data points for any robust image classification task.
2. Class "005.Alaskan_malamute" has 96 data points and class "108.Norwegian_buhund" has only 33 data points. There is a class imbalance problem.
3. All the images are not of the same size (might need image resizing maintaining the aspect ratio).
4. All the images are of the same format – ‘jpg'.

My Initial Thoughts:

There are two ways we could approach the classification task:

1. Train a classification model from scratch
2. Use Transfer Learning

My Rationale:

Training a model from scratch would require a lot of computing power as well as a lot of images per class in order to have some reasonable accuracy. So, I decided to go with the Transfer Learning approach.

Resolving the Class Imbalance Problem:

A class imbalance problem must be handled in-order to ensure that the model isn't biased to a certain class(s) due to presence of more data points in those classes rather in comparison to other classes.

In any Machine Learning Problem, there are generally 2 ways to handle a data imbalance problem: up-sampling or down-sampling.

Since there are so few data points per class, I decided to go for up-sampling rather than down-sampling. I generated some synthetic images using data augmentation. I boosted up the data points per class to match the highest number of data points available in any class in the dataset (ie. To 96/95 data points per class). I didn't just augment one image from a class to make all the synthetic data-points, but I did take care to have a fair share of all the original data points available to make up the synthetic data points.

(ref :: File: Analyze_Data.py :: Method: generate_synthetic_data())

Before jumping into the actual modeling task, I needed to create validation and test datasets to help me measure the efficiency of my model. I wrote up a method to randomly split up the dataset into train/validation/test datasets in the ration 70:20:10.

(ref :: File: Analyze_Data.py :: Method: split_train_validate_test_set())

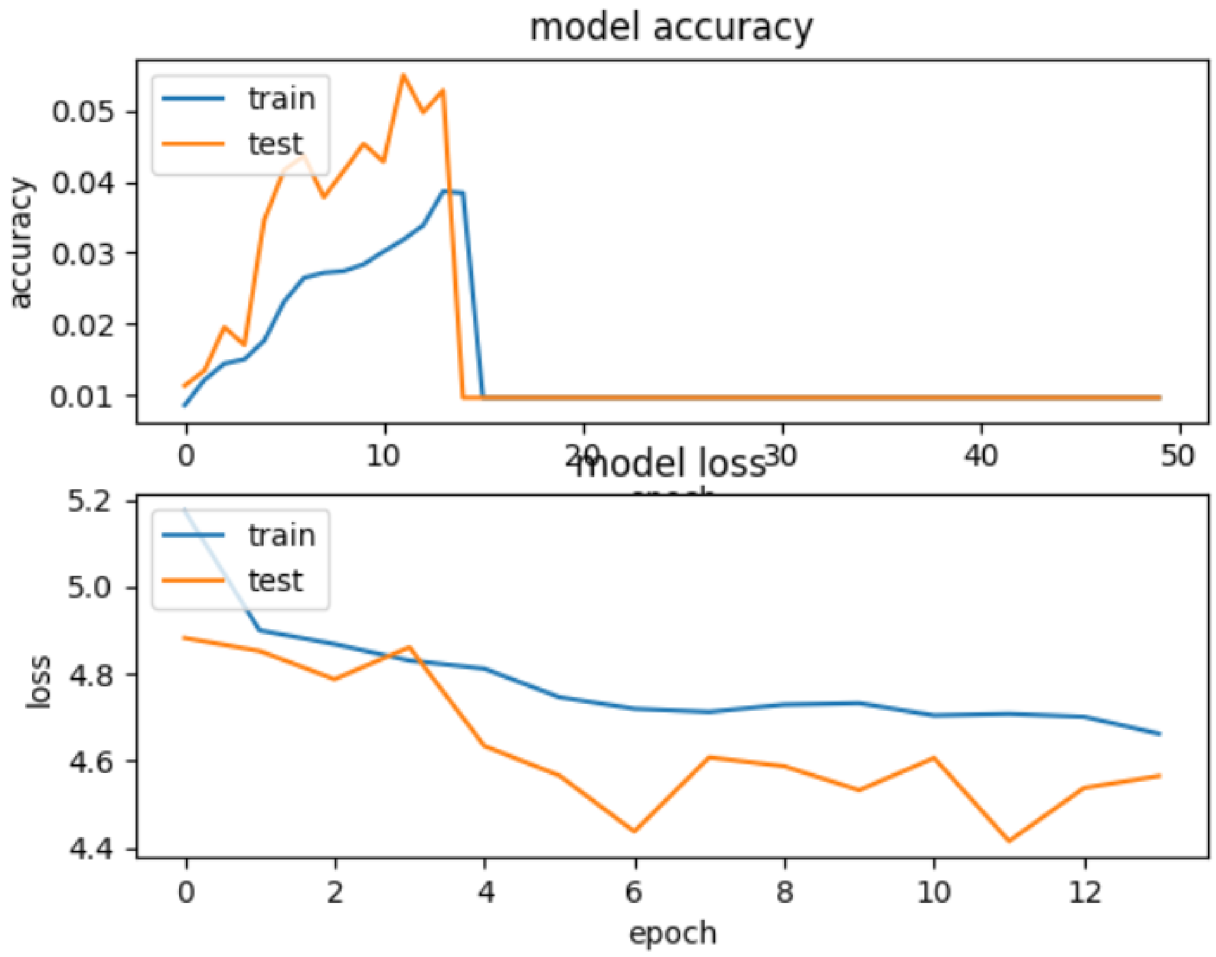
Model creation with Transfer Learning:

Now with my data organized into train/validation/test datasets, I was all set to begin my modeling task.

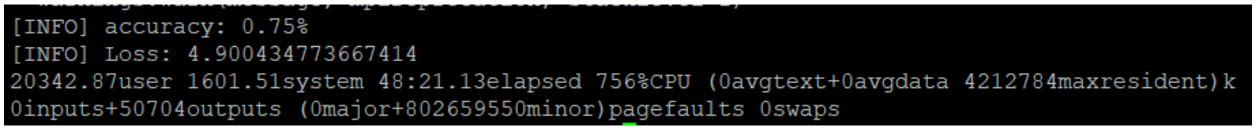
I decided to use the standard pre-trained VGG16 model with its pre-trained weights over ImageNet without the final fully-connected layer. In my past experience, this has been an effective model for Transfer Learning in case of Image classification tasks. I used these layers (all except the last fully-connected layer) to extract the Bottle-Neck features from the small dataset. I then trained a small fully-connected network on those extracted Bottle-Neck features in order to get the classes we need as outputs for the task. I have also tried to augment the data while training in order to reduce the possibility of over-fitting.

(ref:: File: Driver.py :: Method: train_model())

Due to lack of proper infrastructure and un-reliable hotel internet connection, I ran just 2 epochs on my laptop before it actually started heating up. But the same code could be used to run with more epochs and hopefully will give us better performance.



This is the model train history and accuracy and model-loss graphs over different epochs.



```
[INFO] accuracy: 0.75%
[INFO] Loss: 4.900434773667414
20342.87user 1601.51system 48:21.13elapsed 756%CPU (0avgtext+0avgdata 4212784maxresident)k
0inputs+50704outputs (0major+802659550minor)pagefaults 0swaps
```

I was able to achieve a reasonable 75% accuracy on my validate dataset with the above setting.

Business Implications:

As mentioned in the case study this dataset and this task has a huge potential for automating and enhancing the pricing criterion for Pet insurance. To make this into a truly deployable production model we would require a lot more images for each of the different breeds. It is not only sufficient to have different images per breed, but we would also need different images of different ages/genders in order to create a model (or ensemble of models) to detect breed/age/ gender directly from images.

Also the image sizes matter during certain tasks. In case of image classification usually this is not a big problem if the image is centralized to the object in question without any competing objects in the image (eg: a kid with his dog, both dog and kid are competing objects in the image). Having images of the same size, preferably of the size on which the pre-trained model was based on would be a perfect scenario.

There could also be a possibility wherein a user could easily (and happily) upload a couple of photos of his beloved pet and the model could auto-fill all the regular fields in any pet insurer's quotation form like age, mixed-breed or not, gender. Besides the usual auto fills like location (obtainable from the user's current location) etc, the only ones that would remain are the ones pertaining to any pre-existing conditions. This could actually be turned into an easily marketable Data Science solution.

These images could farther be used by the insurance companies to trace any change in the animal's physical outward manifestations like weight gain / broken limp (may be pre-existing condition) during clearing insurance claims.

The prediction function in the solution (ref:: File: Driver.py :: Method: predict()) could be easily be transformed into a backend function for a API layer using something like Flask etc to have a fully connected, end-to-end solution.

Solution Code Files:

Analyze_Data.py – For analyzing the raw data, creating augmented data, resizing the original dataset (not used in final solution), splitting the data into train/validation/test datasets

Driver.py – Extracting Bottleneck features for the transfer learning model, Training the model, Evaluating the model, Predicting the class of a new Data Point (Image)