# Udacity Machine Learning Nanodegree

## Capstone Project - Dog Breed Classifier

Brett Schwarz

December 2020

## Definition

### Project Overview

Machine learning has been around for many years. The phrase was first coined by Arthur Samuel in 1952[1]. However, it hasn't gained much traction until the last decade. This is due to advances in computing power, more sophisticated techniques, more data, and the explosion of use cases.

One of the most exciting use cases for machine learning is image identification and classification. This has many applications and has improved steadily over the last decade. There are many different uses within this domain, including:

- Facial recognition in social media. For example, Facebook's facial recognition.
- Stock photo websites utilize image recognition for categorizing and searching purposes.
- Image searching for finding similar images for businesses. For example, businesses can integrate visual search[2] to enhance their product or service.
- Marketing and advertising campaigns.
- Automation of tasks. For example, Google's address recognition for maps.

[1] https://www.dataversity.net/a-brief-history-of-machine-learning/
[2] https://imagga.com/solutions/visual-search

Machine learning has always fascinated me because it seemed like magic to me. Especially image recognition. Facial recognition was first developed by the U.S. government in the mid-1960s[3]. Before studying Machine Learning, I didn't really understand how this was possible. However, even now that I understand what is involved to implement image recognition, it still fascinates me.

This project will focus on classifying dog breeds, based on input photos. It will also be able to identify if the photo is of a human, and if so, will match the human to the closest dog breed.

One of the most popular algorithms used for image recognition is Convolutional Neural Networks (CNN). CNNs borrowed from nature on how they process images but using a neural network to process the data. The CNN can trace its roots back to the neocognition, which was proposed by Kunihiko Fukushima in 1980. This work was inspired by work from Hubel and Wiesel on the study of vision in mammals[4].

## Problem Statement

The specific problem that this project is solving is to process user-supplied images. It first must detect if the image has a dog or human in it. If there is a dog in the image, then it will determine which breed of dog it belongs to. If there is a human in the image, then the closest dog breed matching the human will be picked. The prediction should achieve a high accuracy score (greater than 60%), and a low log loss.

## Metrics

The model will be evaluated based on the accuracy of the results. The expectation is that the accuracy will be low, especially the benchmark (using a model from scratch) since predicting dog breeds is challenging (even for humans). I will also look at log loss since this can be used to compare the uncertainty of the prediction of the benchmark case against the enhanced case. A lower log loss is expected with the enhanced case.

---

[3]https://www.bloomberg.com/quicktake/facial-recognition#:~:text=Facial%20recognition%20technology%20was%20first,intelligence%20agencies%20and%20the%20military.
[4] https://en.wikipedia.org/wiki/Convolutional_neural_network

# Analysis

## Data Exploration

The data is provided by Udacity, which consist of images. The images fall into two categories:

1. Dog images[5] - there are a total of 8351 dog images. There are datasets for train, test and validation with each having 133 different dog breeds. The training dataset has anywhere from 27 (108.Norwegian_buhund) images, up to 78 (005.Alaskan_malamute) images, so this is not well balanced.
2. Human images[6] - there are a total of 13233 human images. There are 5749 different people represented in those pictures. Some people have multiple images (e.g. George Bush has 530), and others just have one. So, this dataset is not well balanced.

| | **Total Images** | **Unique Images** | **Average Images per Unique Group** |
|---|---|---|---|
| Dog Images | 8351 | 133 | 62.79 |
| Human Images | 13233 | 5749 | 2.30 |
| **Total** | **21584** | **5882** | **3.67** |

Both data sets are needed since we want to detect both dogs and humans in pictures, so we need enough data to train the model. There will be a total of 133 breeds to detect.

[5] https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip

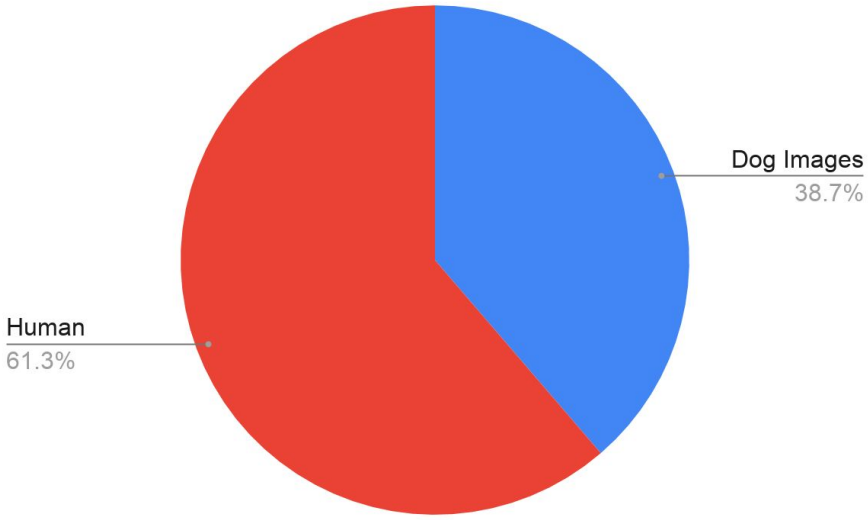
[6] https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/lfw.zip

The human images seem to be roughly the same size. However, some of the images have multiple people in them, and the background varies from bright to dark. Some of the backgrounds don't have much contrast, such as the one above of Zhang Ziyi.

The dog images seem to be a little different in size. Also, some of the images have multiple dogs in them, as well as humans in them.

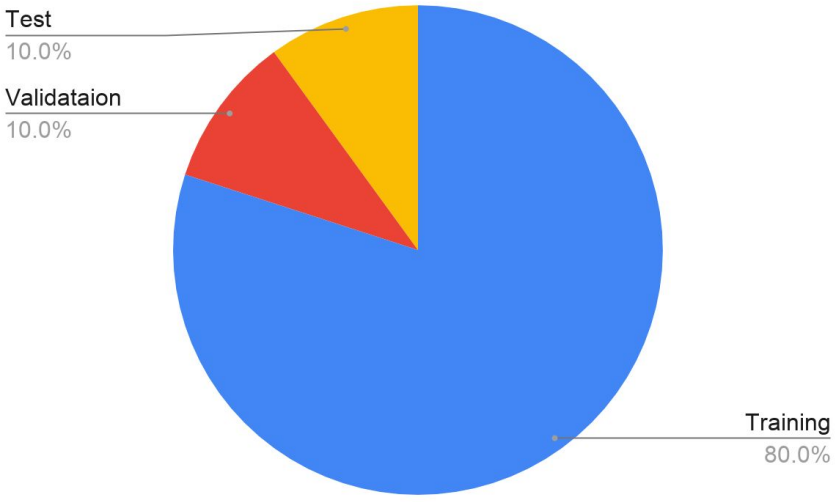
## Exploratory Visualization

The number of images for humans is much more than the number of images for dogs.

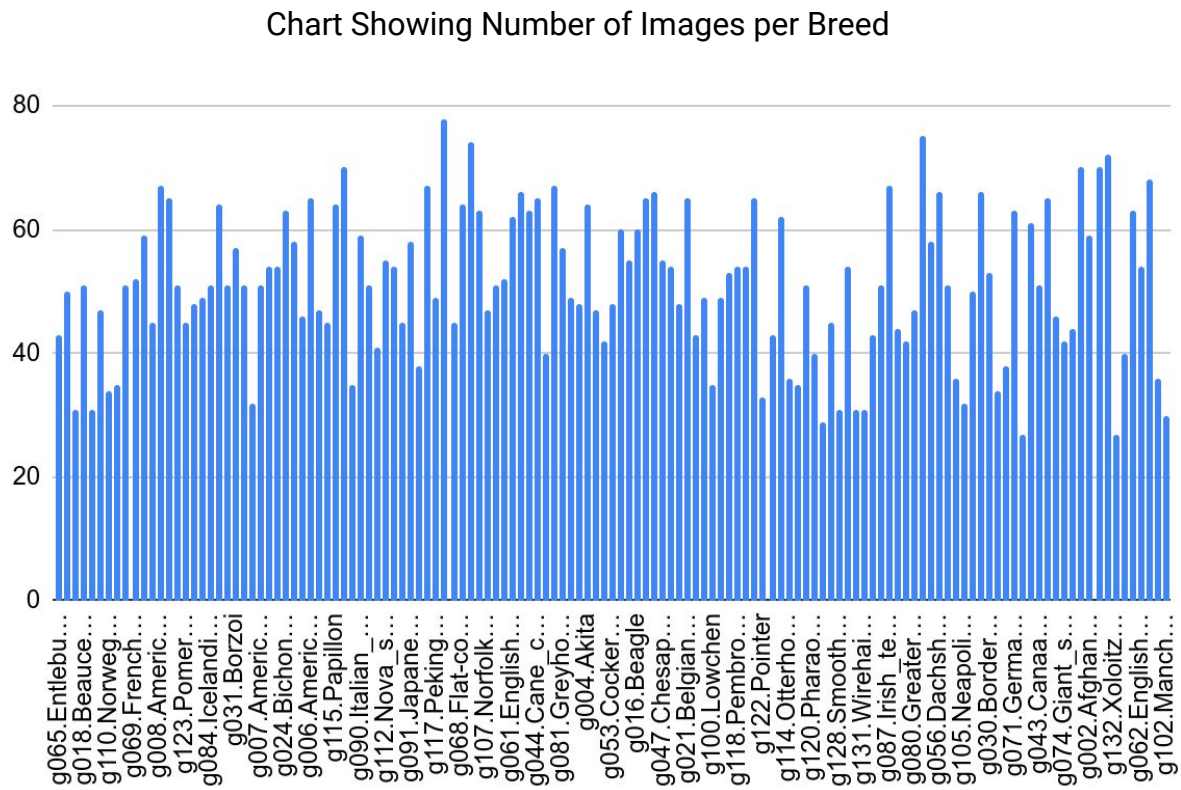



Comparison of the Number of Human Images to Dog Images

For the dog images they are split up between training (6680), validation (835) and test (836) images.




Comparison of the Number of Dog Images for Training, Test and Validation



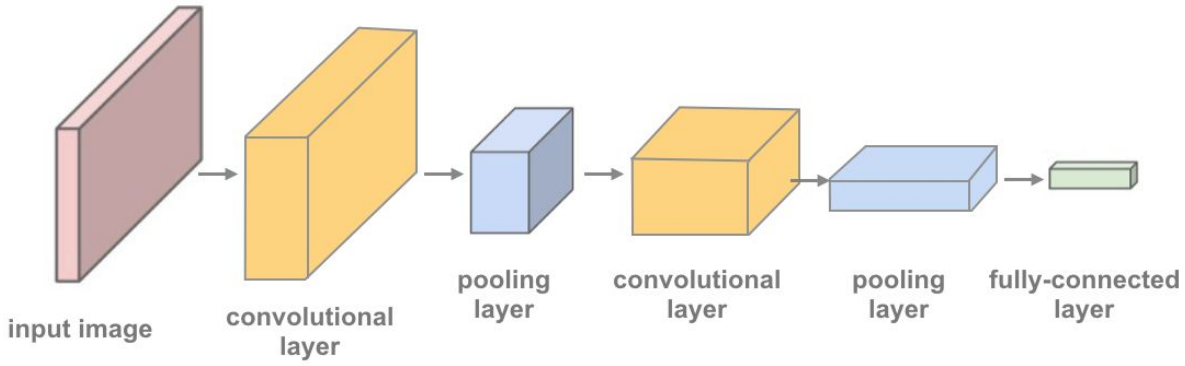

## Algorithms and Techniques

A Convolutional Neural Network (CNN) will be used. CNNs have been shown to do well in image classification problems. Before feeding the image into the model, the image is first checked if a dog is in the image and then if there is a human in the image. For detecting the human images, a haarcascades classifier will be used based on OpenCV's implementation[7]. For detecting the dog images, a pre-trained VGG-16 model will be used[8]. Once the image has been identified, then the CNN model will be used to predict the breed that best matches.

I used 3 convolutional layers, with a max-pooling layer and ReLu layer after each convolutional layer. These are used to avoid overfitting the data. I decided to use the resnet 101 pre-trained model since this involves 101 layers, which seemed more than adequate for this solution. I did try the resnet 50 model as well, with similar results (see end of this paper).

[7] http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html
[8] http://pytorch.org/docs/master/torchvision/models.html

For the training, I used random horizontal flip, center crop, and normalization to put the data into a form that avoids overfitting (gives some randomness to the data).




This is a conceptual diagram of how the solution works. Taken from Udacity lesson[9].

## Benchmark

The benchmark for this project will be the results from a CNN built from scratch. The results will be improved upon after the benchmark by using transfer learning (using a pre-trained model).

From the CNN built from scratch, I achieved an accuracy of 14% and a log loss of 3.794. These numbers are better than a random guess (i.e. 1 out of 133), but they can be significantly improved upon.

---

9 https://classroom.udacity.com/nanodegrees/nd009t/parts/e8261bb7-c4bb-4c9c-8eae-2347c5aa92d2/modules/d40229df-354e-4792-918e-544d1e42cf99/lessons/807590ea-abd5-4581-b91d-9eede9a0aad2/concepts/3d87c586-5dae-4eb4-85a1-01f3151a025f

# Methodology

## Data Preprocessing

All of the images were resized into squares of dimension 224. This was the minimum size based on the PyTorch documentation. The images were also normalized per the PyTorch documentation[10]. For the training images, I also did a random resize, center crop, and random horizontal flip in order to randomize the data better, to avoid overfitting.

## Implementation

I am using a CNN with three convolutional layers. After each layer, I use a MaxPool layer of 2 to reduce the image size. In between each layer, I add a ReLu function. After the convolutional layers, I flatten the results and then apply two fully connected layers to bring the output to 133, which represents the number of different dog breeds. Also, a dropout function is used to minimize overfitting.


[10] https://pytorch.org/docs/stable/torchvision/models.html

### Refinement

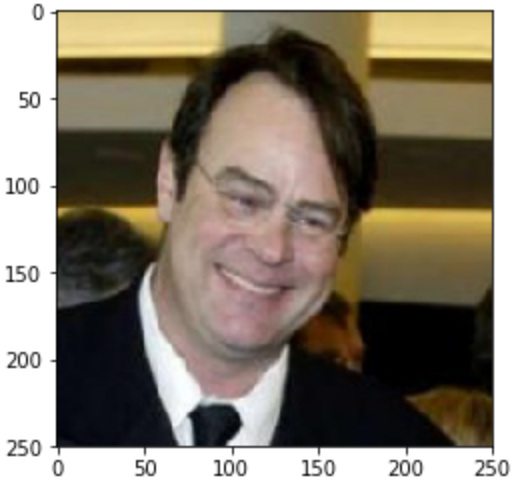
The original benchmark solution didn't have a good accuracy score, and the log loss was not good as well. Instead of using a CNN from scratch, a pre-trained model was used, since these have already been tested and optimized. Using a pre-trained model improved accuracy significantly, from 14% to 85%, even with a reduction in the number of epochs. The log loss was reduced as well, from 3.794 to .5828.

## Results

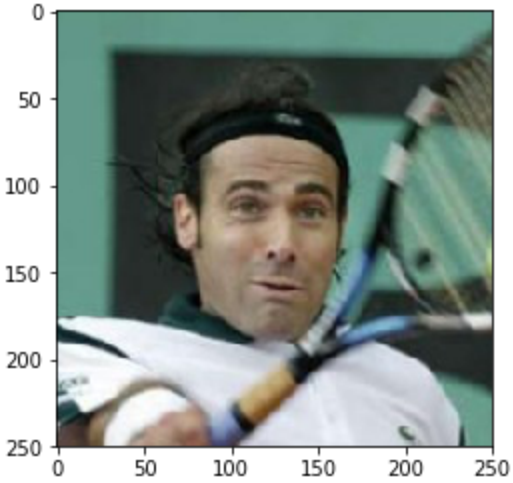
### Model Evaluation and Validation

The CNN using the pre-trained model resnet101, did fairly well with an accuracy of 85% over five epochs, with a log loss of 0.5825. Over 10 epochs it achieved an accuracy of 85% and log loss of 0.4777. After analyzing some random samples, the solution both selected human faces and dog faces accurately and selected the dog breed accurately. I can not give a good assessment of the accuracy of the dog breed in the case of a human face, since that would be subjective.

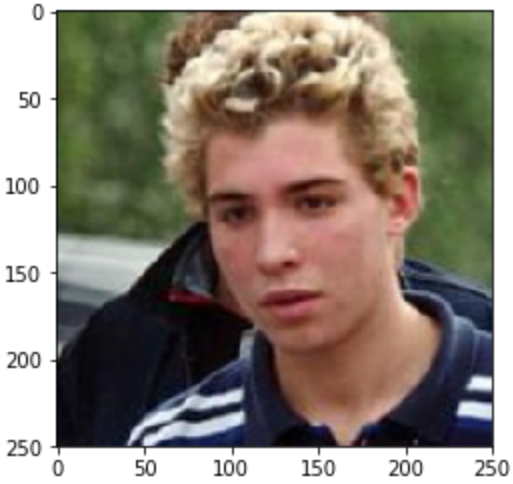
There is a human in the image. The predicted breed of the human is a:  Beagle



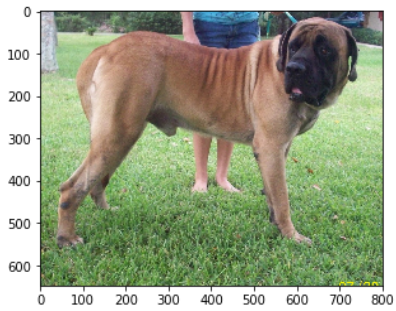
There is a human in the image. The predicted breed of the human is a:  Brittany



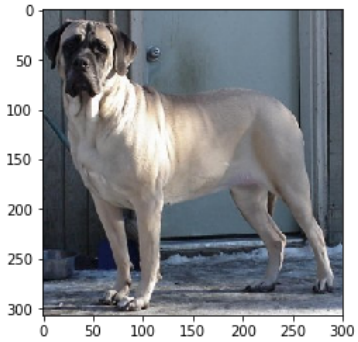

There is a human in the image. The predicted breed of the human is a:  Black russian terrier



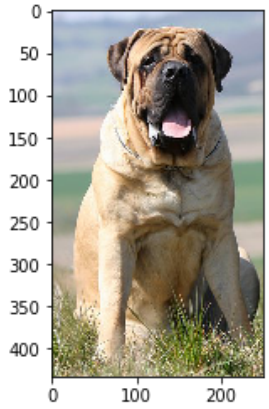
There is a dog in the image. The predicted breed is a:  Mastiff




There is a dog in the image. The predicted breed is a:  Mastiff




There is a dog in the image. The predicted breed is a:  Bullmastiff



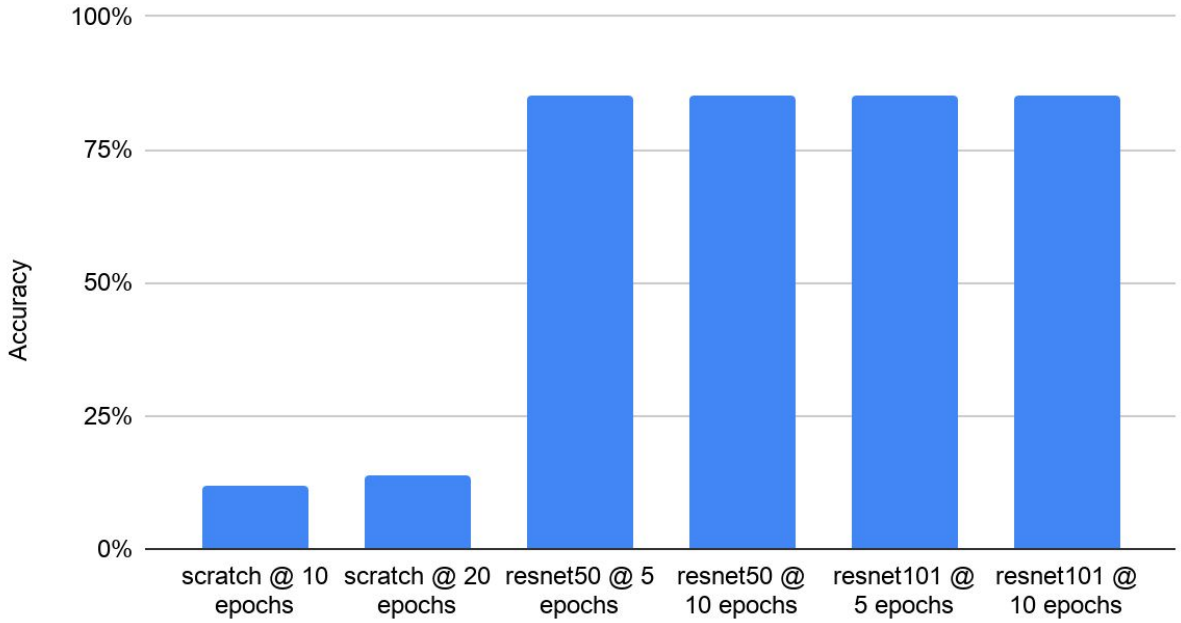
## Justification

The use of transfer learning was actually better than I expected, especially given that the model from scratch was so poor. I am surprised how much improvement that transfer learning had on the accuracy. The accuracy went from 14% to 85%, a 6 times improvement. The log loss also went from 3.794 to .5825, a 6.5 times reduction. Given the difference in the effectiveness of the solution, using the pre-trained model is a valid and useful solution. I also compared the resnet50 pre-trained model, and it gave similar results to resnet101. Here are results based on the different runs:

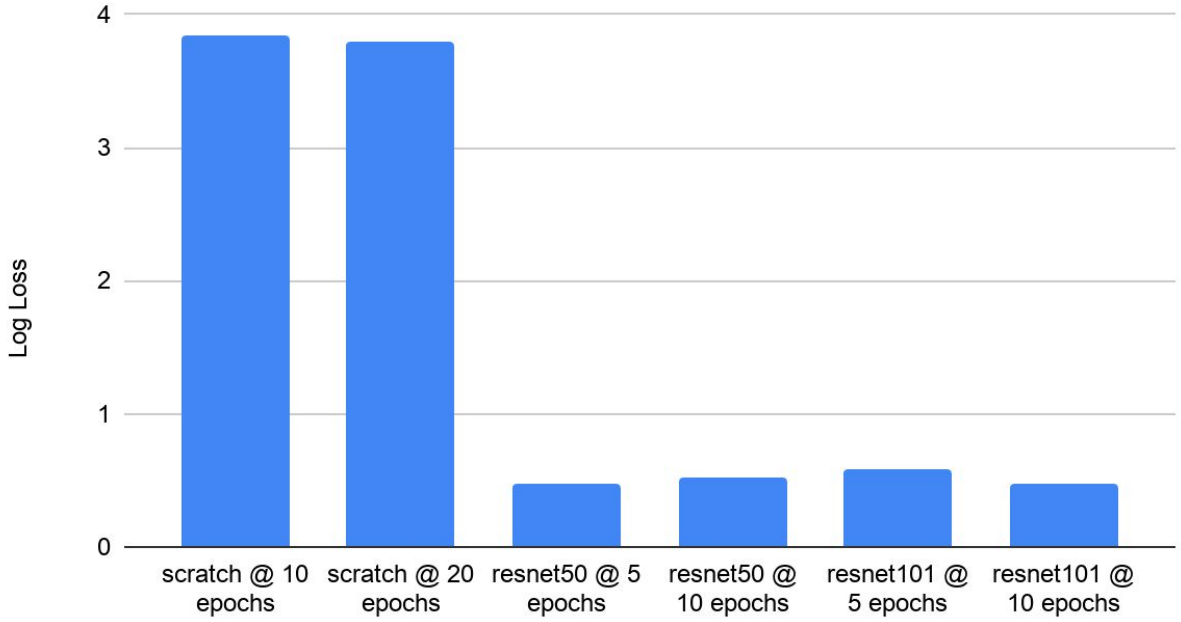
Accuracy and Log Loss Across Runs

| | **Accuracy** | **Log Loss** |
|---|---|---|
| **scratch @ 10 epochs** | 12% | 3.8414 |
| **scratch @ 20 epochs** | 14% | 3.794 |
| **resnet50 @ 5 epochs** | 85% | 0.4795 |
| **resnet50 @ 10 epochs** | 85% | 0.5166 |
| **resnet101 @ 5 epochs** | 85% | 0.5825 |
| **resnet101 @ 10 epochs** | 85% | 0.4777 |

## Accuracy




## Log Loss



As you can see from the data, there was a significant improvement moving to transfer learning in both the accuracy score as well as the log loss. However, the difference between the resnet50 and the resnet 101 pre-trained models were insignificant, even when run from 5 epochs to 10 epochs.

The results may improve by doing any of the following:

- Adding more training data, especially if the data becomes more balanced between the breeds.
- Do some more data augmentation, to get the data in a better input state. For example, could possibly try doing a random flip of the images, in addition to the other data augmentation.
- Could experiment with other pre-trained models as well. I used resnet50 and resnet101, but could try others to see how they perform.