CS 82 Final Project: Dog Breed Detection System

Sharjil Khan

1. **Introduction & Motivation**

Classifying dog breeds correctly allows adoption agencies to look at a dog’s likely traits based on their breed. This allows them to explore the dog’s differentiating personality as well, so that they can provide a complete picture to possible adopters. Further, adoption agencies often work with volunteers who may not specialize in breed identification and having a program that can evaluate a dog’s picture to help determine a breed can allow the volunteers to help in the dog’s evaluation.

Detecting the breed of a dog from its visual appearance is not an easy task. The task is made even more difficult by the fact that most dogs are a mixture of different breeds. Having a tool to at least narrow down to a few breeds when trying to determine the breed of a dog would be of great help to many agencies that deal with dogs.

1. **Scope of the Project**

Given the time limitations to implement this project, I have narrowed down the scope of this problem to a set of simpler more manageable problems. I will first try to train models to differentiate between dog breeds that are more easily distinguishable. For example, breeds with different colors or a major size difference. I will gradually try to add more dogs and see how we can adapt the models as the problems become more difficult.

The final goal of this project is to differentiate between 8 dog breeds with an accuracy level which would be significantly better than blindingly assigning the dog to a specific breed. I want to be able to get to a point where given enough training data, I could conclude that the models would be able to differentiate between the dog breeds.

I believe Convolutional Neural Networks (CNN) would be the ideal model to use for this problem. Because CNNs are in general great for pattern recognition in images. However, given the reduced scope of the problem, I also intend to try out another simple color pixel counting method and other kinds of classifiers such as LDA or KNN to see if we could achieve similar results that a CNN model would.

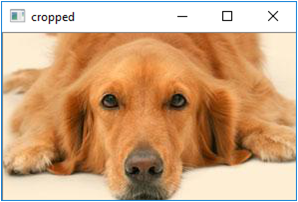
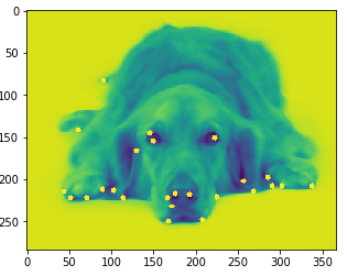
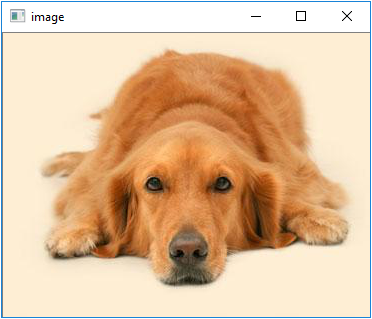
1. **Data Used**

The data used for the project comes from Stanford Dogs Dataset**:** <http://vision.stanford.edu/aditya86/ImageNetDogs/main.html>.

The data is structured into directories of dog images. The images are in sub folders based on their breeds. As a first step, I split the data into a training, validation and test set and use Scikit-learns data generator modules to feed the data into the models in batches. The pictures in the dataset. The pictures have different pixel counts that make it a challenge to compare them against each other. The dogs we want to detect are also part of a larger environment, which introduces noise in the data. I tried to use some libraries in python to try to address some of these issues and focus on the more important parts of the images to improve accuracy scores. These methods are described in detail in the following sections.

1. **Steps taken to Prepare the Data**

An initial exploration of the images, made it clear that the data is very noise. In a lot of the images, the dogs are occupying a small part of the overall image which contains a lot of other objects in the environment. In some of the pictures, there are even objects in front of the dog, partially covering the dogs. For example even in a simple picture like the one shown in Figure 1, more than 50% of the image is occupied by the environment which gives us no information about the dogs breed. I used an open source python module, OpenCV to process the image. The module uses a feature selection function to zoom in on more informative parts of the image. This cuts out the parts that reveal no information about the dog breeds and makes it much easier to learn about the specific dog breed traits with less noise in the training set. I found applying a simple open source filter like this greatly increased the accuracy results.

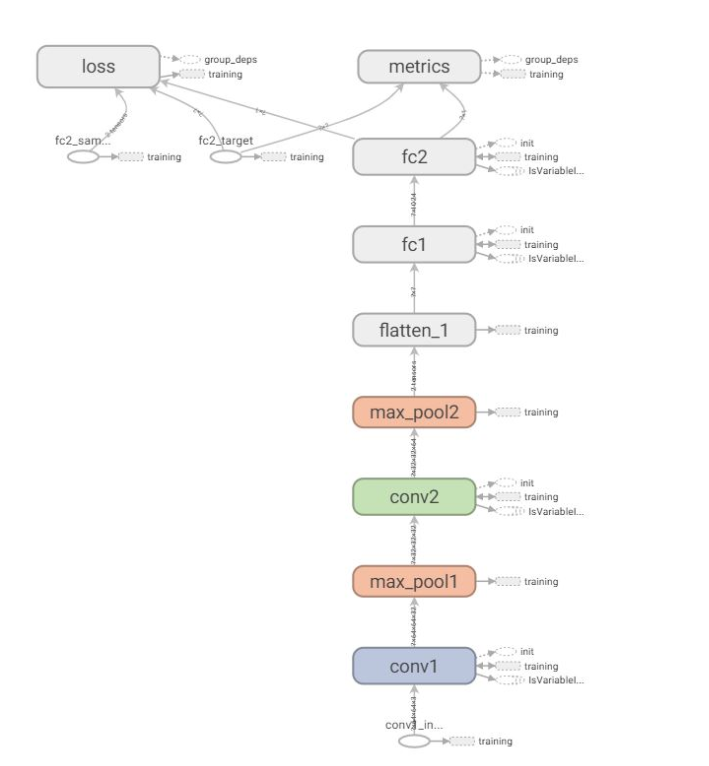


**Figure 1** In the left image, the dog is occupying around 50% of image area. The middle image shows how the key features selected by the feature selection function from OpenCV. The image is then cut using the co-ordinates from the feature selection function which results in the dog occupying about 75% of the image on the right.

After applying the above-mentioned filters, I split the images into a training, validation and test set to be used by Python’s data generator module that can feed the images in batches to the CNN model for training and testing. Apart from this approach, I also used a simple color count strategy to use simpler models such as KNN which is described in Section 5.1.

1. **Models and Approaches**
   1. **Convolutional Neural Network:**

I used a simple Convolutional Neural Network with 2 convolutional layers along with a couple of max pooling layer to narrow down the outputs as we get closer to the output layer. I used to fully connected layers at the end to output a categorical classification result using a SoftMax activation function. I wanted to keep the design simple, so I could easily see the effect changing a parameter which would then give more insight into how to get the best results out of it.



**Figure 2** TensorBoard representation of the Neural Network

My eventual goal was to move to a Transfer Learning based model and therefore I wanted to keep the CNN Model as simple as possible. I intended to eventually add this neural model to a pre-trained model provided by Keras so I wanted to have a clear understanding of the effect of each of its components on the output. But this very simple model performed very well on recognizing dog breeds as long as the number of classes(breeds) where not too high. The results section describes the accuracy rates in more detail.

* 1. **Transfer Learning**

I tried several pretrained models such as ResNet40, Xception and InceptionResNetV2 provided by Keras. After some initial testing, it was obvious that Xception trained on the imagenet dataset was getting the best test results. So I focused on the Xception model as my main pre-trained model to build my model with pre-trained weights.

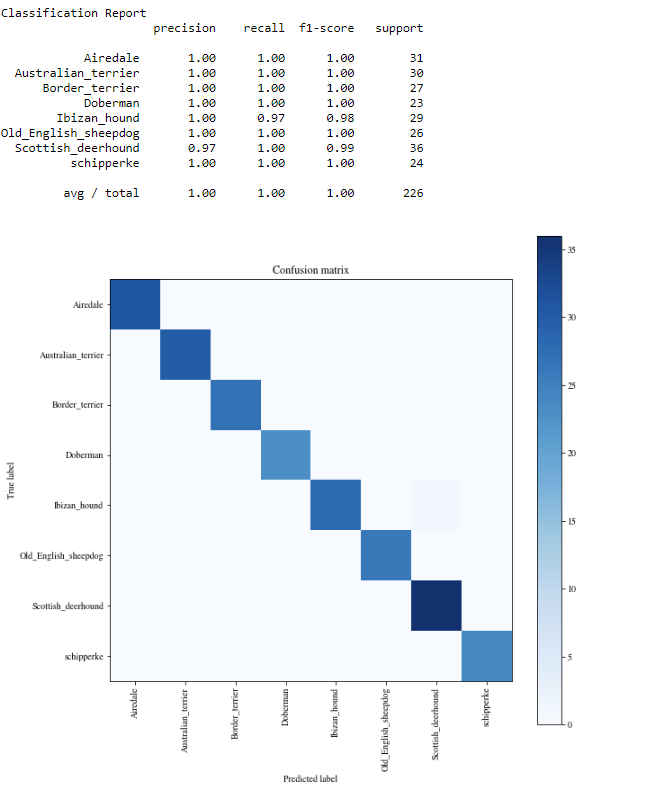
I also wanted to tune the Xception so I opened up about 40 layers to be trainable during my training process and kept the rest of the weights frozen. I did not add the top layer from the Xception model and replaced it with a dense fully connected layer of my own with the number of classes I wanted to classify as the final output of the model.

Through some trial and error, I saw that adding a few convolutional layers

1. **Results**
   1. **Convolutional Neural Network**

Using a Convolutional Neural Network model which was described in Section 5.2, the accuracy score for classifying between 8 dog breeds was almost 100%. The Results below demonstrate that.

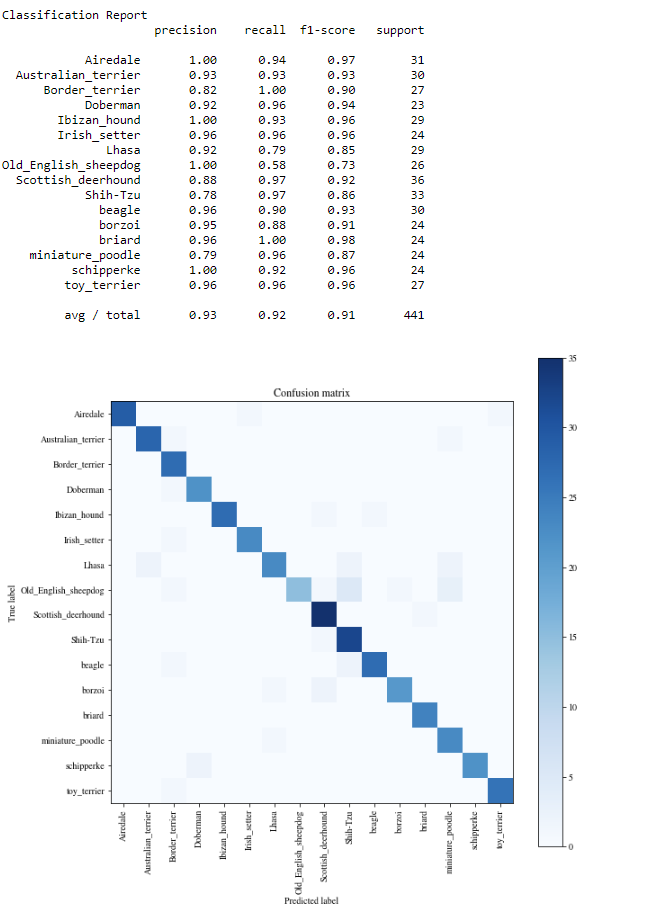
**Test Accuracy with 8 dog Breeds: 99.6%**



**Figure: 3** Results showing near perfect precision, recall and accuracy for 8 dog breeds. The confusion matrix also shows nice deep line along the diagonal.

With such great accuracy scores, I decided to push this model to tackle a more difficult task of classifying between 16 dog breeds. Without any change to the model parameters, the accuracy scores on the test set was again surprisingly good with average accuracy levels at over 90%.

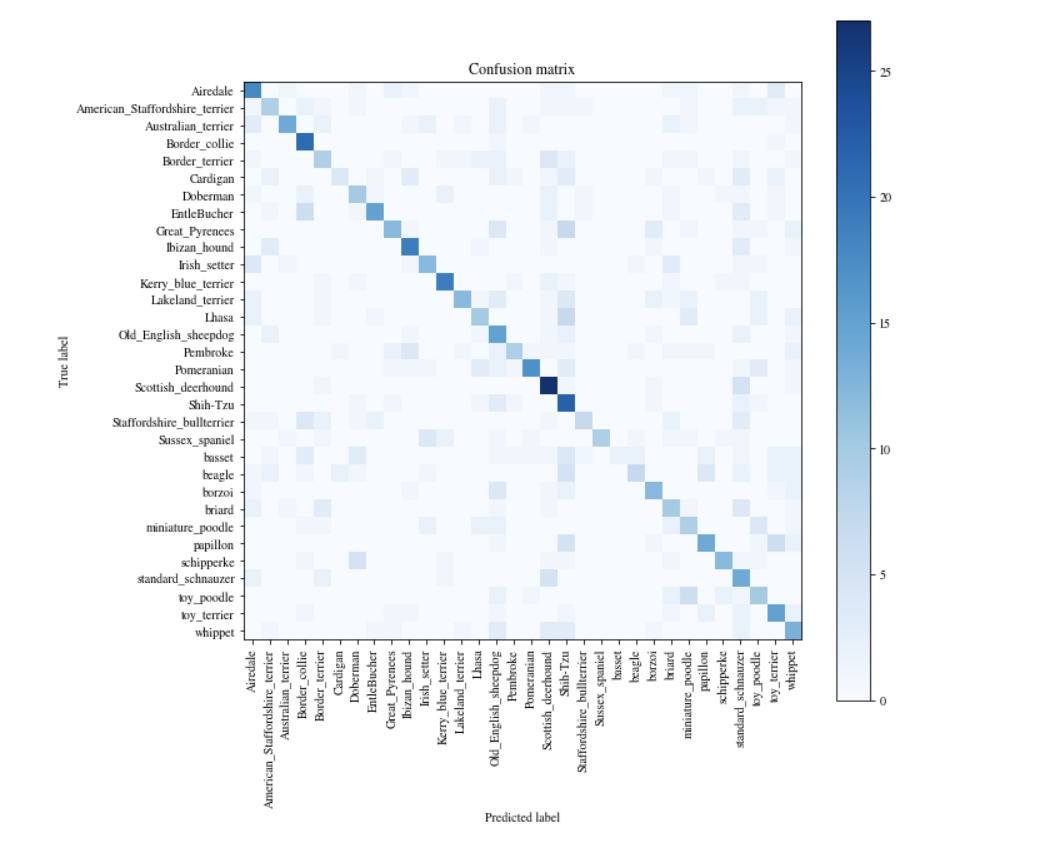
**Test Accuracy with 16 dog Breeds: 91.17%**



**Figure: 4** Classification Report for classifying 16 dog breeds using CNN

Then I doubled the number of dog breeds to 32 making it a much more difficult problem to handle for this relatively simple CNN model. This time the accuracy scores dropped well below 90% and the best accuracy I was able to achieve was about 46.12%. Since the results in **Figure 4** fell far below 90% and I have only added 32 dog breeds in the mix, I decided to stop tuning this CNN model anymore and move to a pre-trained model, that I could tune to my dataset to achieve better scores .The results of the transfer learning approach is described in the next section.

**Test Accuracy with 32 dog Breeds: 46.12 %**

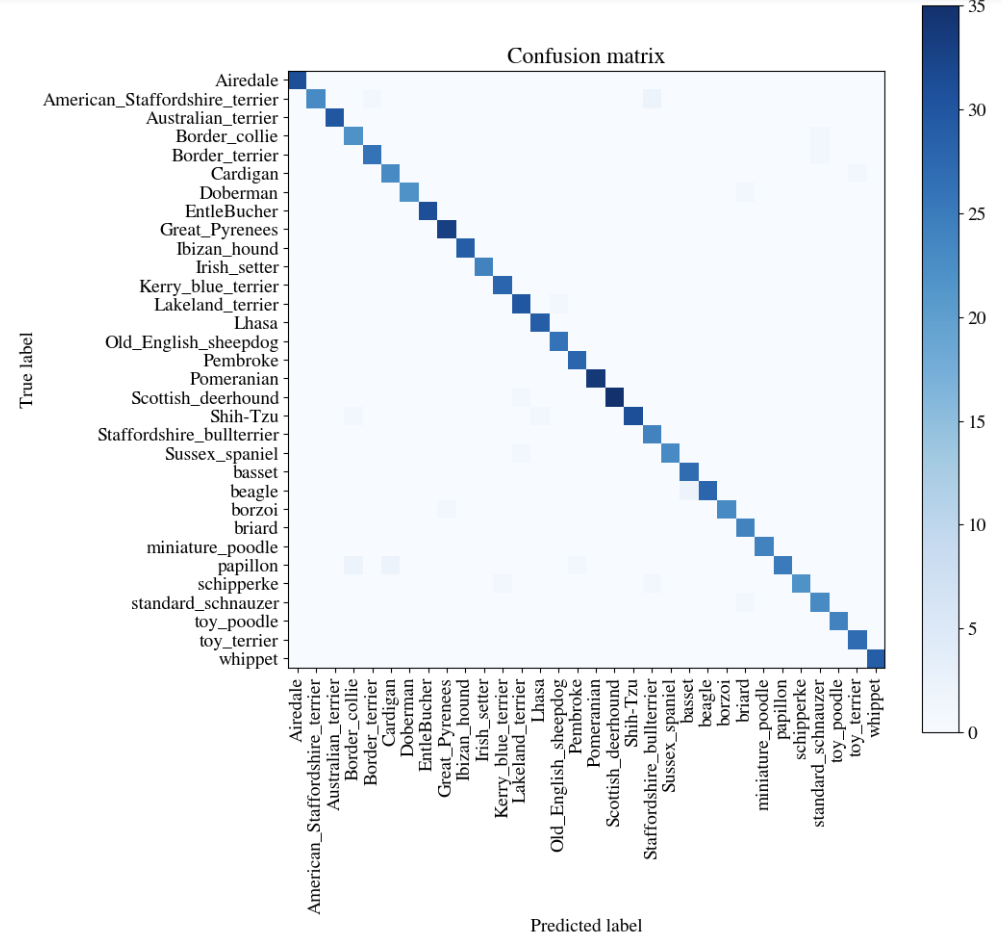


**Figure: 4** Confusion Matrix showing results of classification between 32 dog breeds

* 1. **Transfer Learning**

I used a transfer learning approach as described in Section 5.3 and was able to improve the results of the simple CNN model significantly on the problem of detecting between 32 dog breeds.

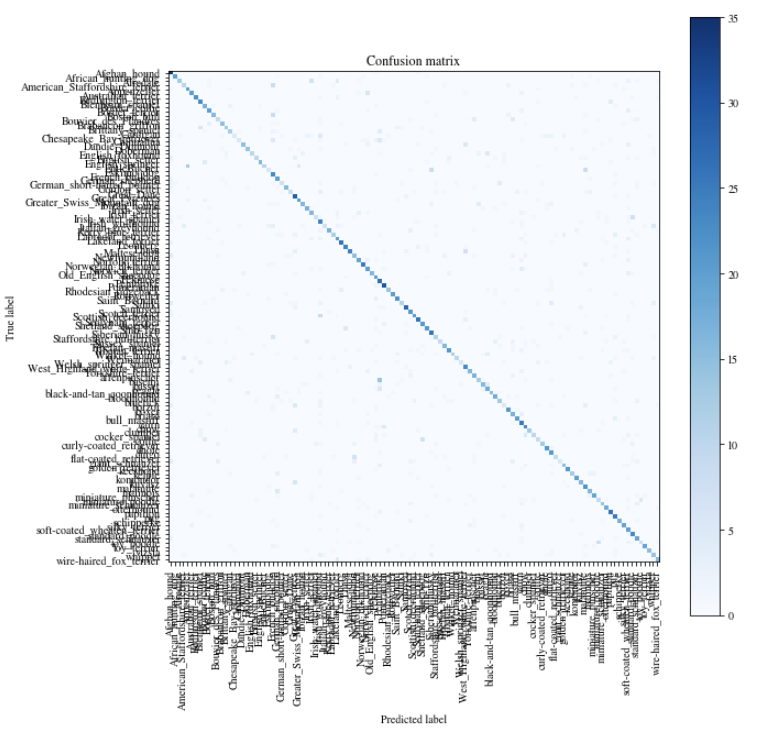
**Test Accuracy with 32 dog Breeds : 97.12 %**



**Figure: 4** Confusion Matrix showing results of classification between 32 dog breeds using a pre-trained model as a base.

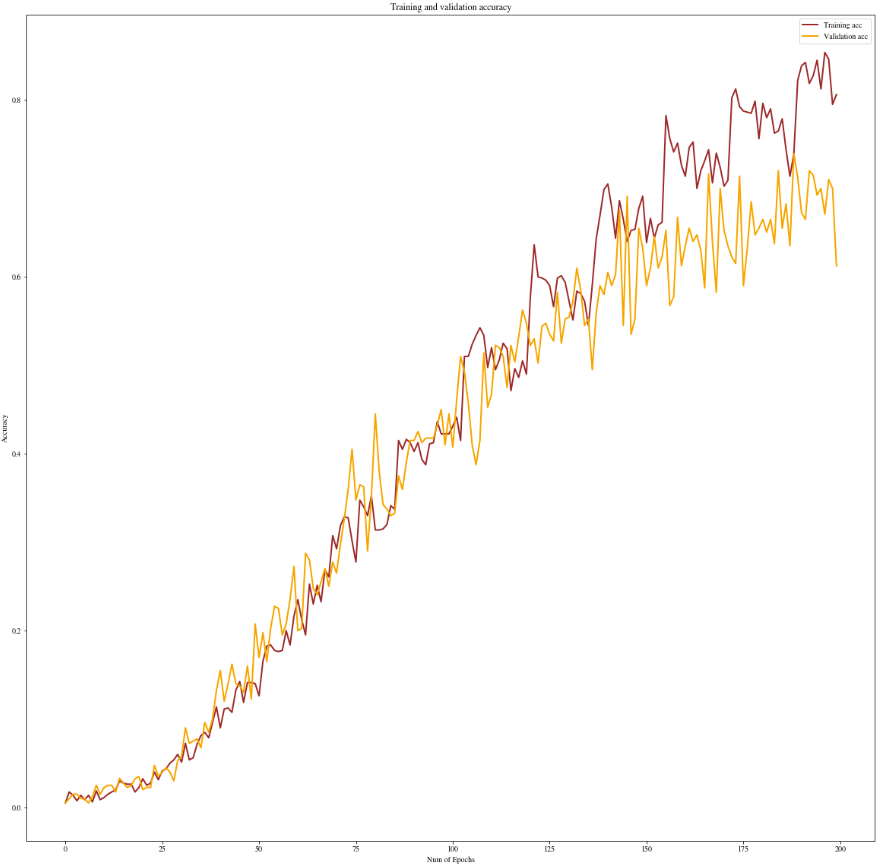
With the partially pre-trained model performing so well on the classifying between 32 breeds, I decided to move forward rapidly and try out a model with 115 dog breeds. I increased the number of epochs to train on and also increased the number of layers in my convolutional layers to be able to capture more distinct features in order to differentiate between large numbers of dog breeds. It achieved a descent accuracy score of 58.75% with some room for improvement.

**Test Accuracy with 115 dog Breeds: 97.12 %**



**Figure: 4** Confusion Matrix showing results of classification between 115 dog breeds using a pre-trained model as a base.

In the confusion matrix above, we are able to see a nice diagonal deep strip showing a lot of the dogs were being classified correctly in the test dataset. But I believed the 58 % accuracy rate could definitely be improved upon. A look at the training vs validation scores during training showed the model was beginning to overfit as shown in Figure X below.



Training score & Validation scores diverging showing signs of overfit model.

**Figure: 5** Training vs Validation scores during training showing signs of overfitting

So I add a batch normalization layer and a drop out layer to try to reduce the overfit and decreased the number of layers in the pre-trained model that I was allowing the new data to train. Having done these I also increased the number of epochs I was training with because I expect the model to start to over fit at a much later time now that I have added the measures to reduce the level of overfitting.

With the new setup I was able to achieve much better results with accuracy rates of around X percent. The figure below shows the results of running the model on test data for 115 breeds of dogs.

1. **Conclusion**

This problem is particularly difficult because we are trying to differentiate between pictures that are very similar in nature. All the pictures have dogs in them which in general have very similar features with very subtle difference between the breeds. Having said that, I was very surprised by how accurately even a very simple Convolutional Neural Network was able to detect different breeds with very little training. It shows that convolutional Neural Networks are very good at detecting patterns in images.

But for the more difficult problem of detecting all dog breeds which would number in the hundreds we need much larger models with more layers and parameters. We would also need a lot of training data to train such a model. In those cases I was able to demonstrate that transfer learning is probably the best way to solve it where the model is already pre-trained and we don’t have to retrain all the parameters ourselves. And the improvement on the results with transfer learning demonstrates that. I will definitely continue to build on this and hopefully manage to get to a point where the breed detection system will be able to detect any dog from just one image of the dog. I believe if I get a good enough model to transfer parameters from and train it with enough additional dog images, that goal can be achieved.