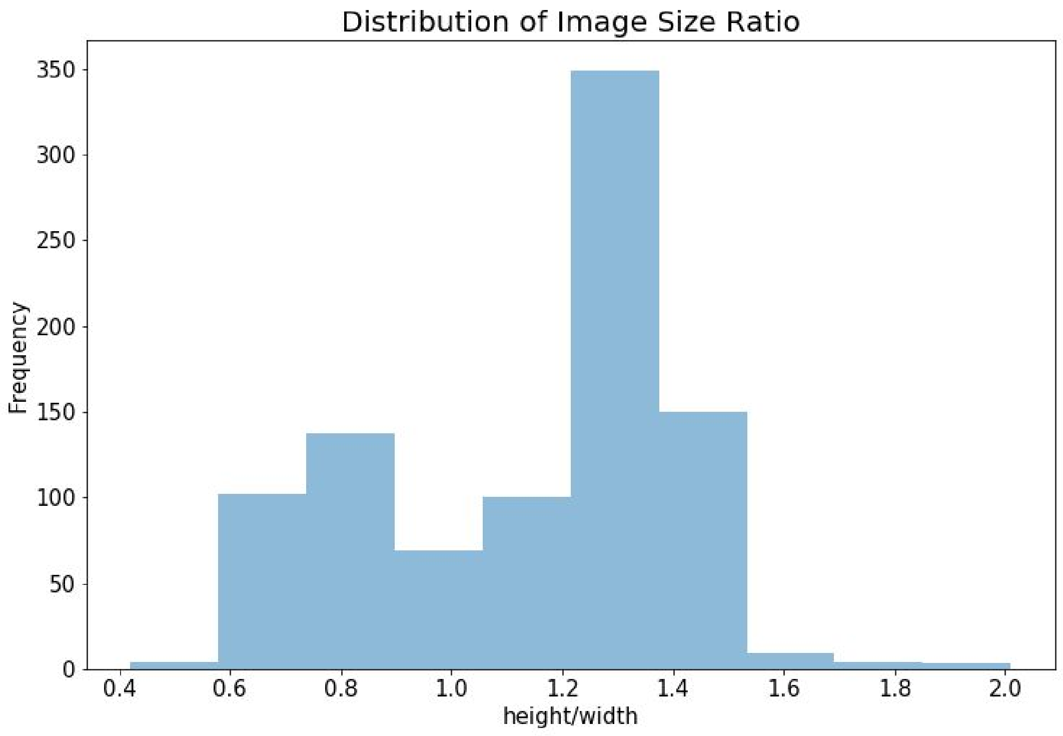
Group #36 Milestone 3: Dog Breed Classification

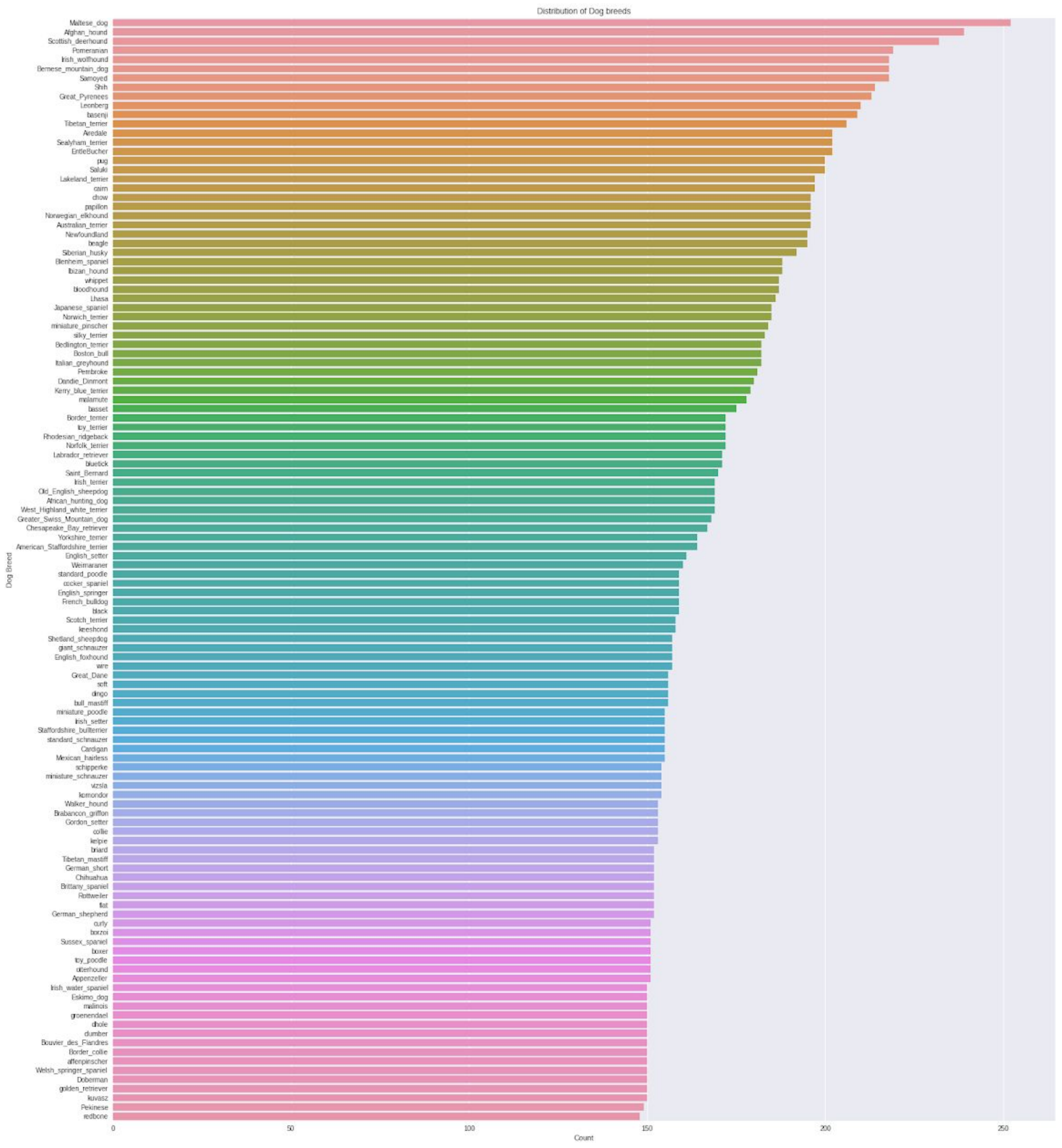
Eimy Bonilla, Ada Shaw, David Wei

Description of Data:

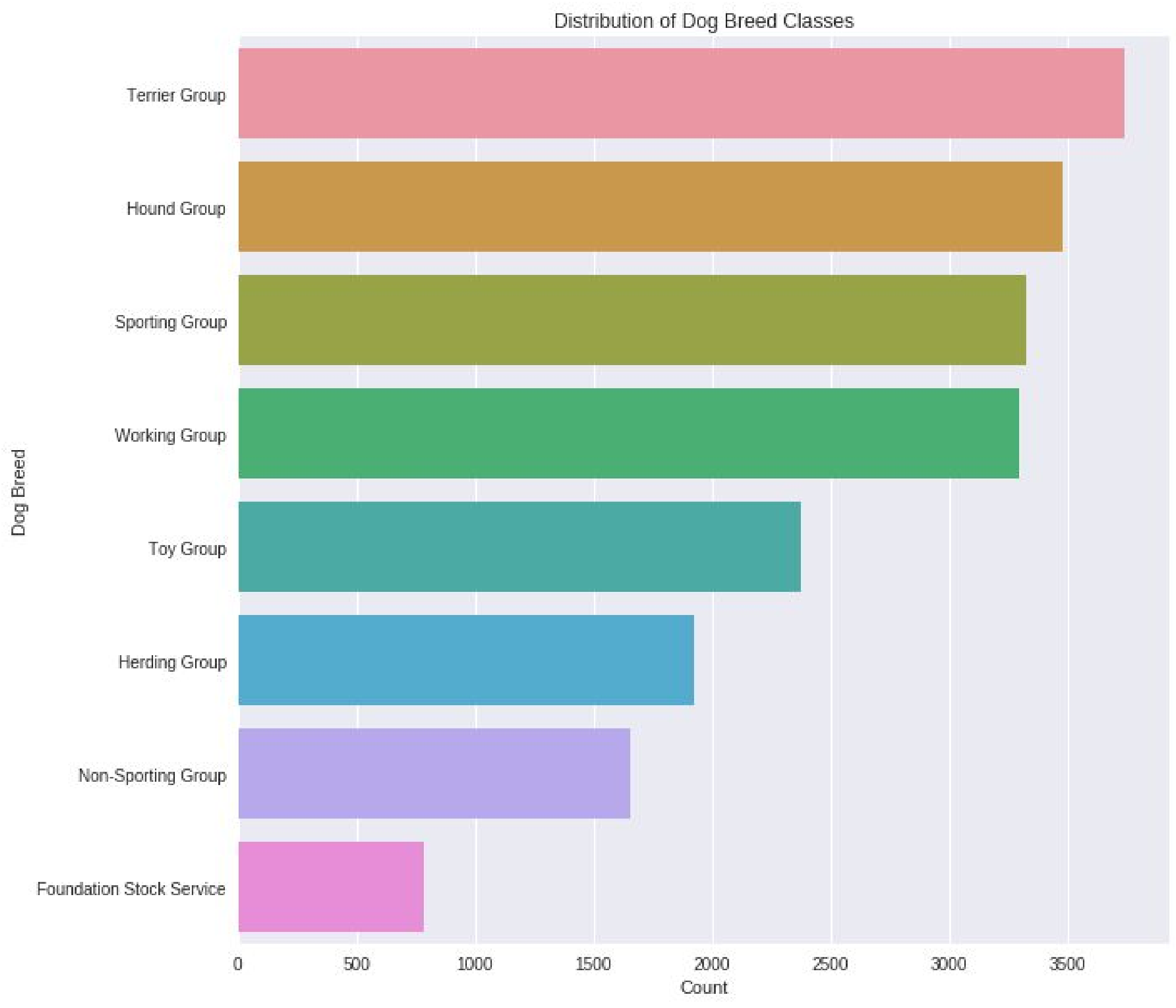
The data that we are classifying is 20,580 dog images of 120 breeds from the Stanford Dogs Dataset. Overall this leaves ~150-250 dogs per breed to train the model. Because of the low amount of images per breed and thus possibly insufficient data for effective training, we coagulate the 120 breeds into 8 breed classes. We fit a CNN model to our data. To optimize the performance of the model, we use data augmentation techniques. The following plots show some of the EDA that we used to analyze the images. The first one shows the distribution of image size ratio (height/width). The range of the size ratio is from 0.4-2.0 with most images having ratios above or below 1. These wide ranges of height and width imply that rescaling all the images to a size of 250x250 could possibly lose the dog.



The following image shows the distribution of dogs across breeds. It shows how there is a large difference between the most common dog and the least common dog (~100 images) and the most common dog only has ~250 images.



We use breed classes instead of breeds to classify our dogs, leaving us with 8 breed classes and at least 700 dogs per class. There are a wide variety of dogs that qualify as terrier dogs.



<u>Original Project Question:</u>

How well can our models classify dog breeds? Specifically, how does the accuracy of the classification change with different model implementations?

<u>Modified Project Question:</u>

How can we optimize the model to classify dog breed classes? How does the classification change when we experiment with data augmentation or different scaling methods?

<u>Current Model Description:</u>

We use a CNN because convolution layers allow for fewer parameters per number of layers compared to fully connected layers (dense layers). ANN's can become to be very slow with images as the image size increases and thus will limit the number of layers or nodes per layer. Rather than assigning a parameter per every two neurons, Convolutional Neural networks assigns a parameter to a block or group of neurons. In CNN's, instead of having each neuron connect to all pixels, some connections between pixels and neurons are "convoluted" rather than direct, i.e. they are through another pixel. Spatially correlated pixels will be grouped together which makes more sense because pixels closer together will have more in common than ones further away (e.g. a cloud in the sky will have similar pixels than the ground pixels). CNN's are efficient in time and memory requirements, a must for images which usually are memory intensive.

We scale the input training data by standardizing per-channel. We by subtract the per-channel means of the final training set and divide by the per-channel standard deviation of the whole dataset.

To get an equal distribution of breeds within the breed classes in the training set of the model, we use 100 images per breed, however we find that with data augmentations, the model is especially slow with around 10 minutes per epoch. This is already with downscaling the images to be 64x64 versus 250x250. Because Keras automatically parallelizes its training processes, we need to increase computational resources by adding more cores.

Next Steps:

Our model performance is very poor with a 15% training accuracy and a 11% validation accuracy. This means that the model is just as good as random guessing. We need to explore more model optimizations, including trying resnet instead of CNN. We have implemented a simple resnet using the keras resnet50 function and achieved 95% training accuracy and ~25% validation. The resnet50 is a 50 layer residual neural network.