Springboard Data Science Intensive Program – Capstone Project #2

Who’s a good dog?

An exploration and assessment of using Convolutional Neural Networks to classify dog breeds from images

Garrick Chu (2018)

Table of Contents

In memory of Pepsi (Pembroke Welsh Corgi, 4/22/2007 – 4/6/2018).

1. Introduction

1a. Motivation/Background

My initial intention was to learn and familiarize myself with neural networks and understand why these methods have been at the forefront of artificial intelligence in recent years. However, I couldn't convince myself of a real-world problem or application.

About 2 weeks into this process, I went to a nearby park for a morning walk where I came across a young husky pup running around aimlessly in an open field. Soon after, I noticed the husky would run towards other people to investigate and then dart away. It finally approached me and, while petting in a manner to ease anxiety, I realized it wasn't wearing a collar. Taking consideration that Chinese New Year had just passed, I figured this husky pup got scared from the weekend barrage of fireworks and ran off. With some quick thinking, I snapped a few pics and posted on Facebook with as many details as I could. I was able to reunite the dog with the owner in less than 3 hours.

However, I couldn't help but wonder how many other lost dogs and their families had not been as lucky as this husky pup. How many tragedies and heartbreaks could have been avoided by empowering a stranger with knowledge they otherwise wouldn't have simply by capturing a photo?

I propose an application that can take images of a dog and gives its best estimation of the dog breed, hopefully exceeding the average human level of accuracy. At 167 breeds recognized by the American Kennel club, even canine enthusiasts could struggle to correctly identify the majority of breeds from images. I believe an application such as this will come to the aid of municipal animal control services and good Samaritans recognizing an opportunity to reunite a lost dog with their family.

1b. Business Use Case

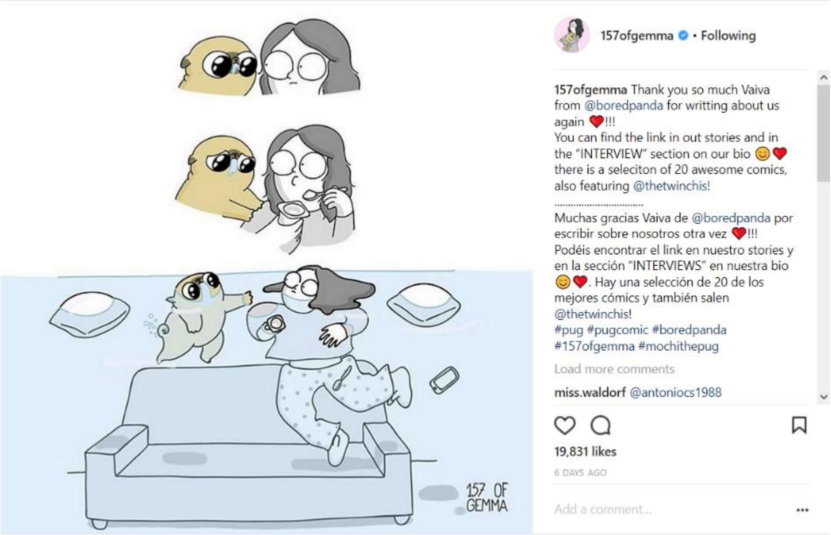
There are commercial and business-use cases for such an application/product as well. With social media channels thriving off image-driven content (like Instagram), advertisers such as pet food, toy makers, lifestyle brands and social media influencers could benefit from being able to reach new audience members that otherwise would not have been identified from their text information only. More specifically, users that have pictures of their dog on their account but with no associated hashtags or text caption.

This may seem like a frivolous endeavor until one begins to grasp the economic scale of social media and pets (with dogs dominating the market). Here are some quick figures to help appreciate the massive scale of this market:

- America's pet industry is estimated around $69.51 Billion as of 2017 and estimated to grow by 3.7% to $72.13 Billion. [1]
- Each sponsored post can command around $2,000. [2]
- Sponsored posts by high-profile pets can range from $10,000-$15,000 per post. [4]
- Grumpy Cat's Net Worth: ~$100 million. [3]
- Boo the Pomeranian has 17 million followers on Facebook and earns $1 million USD per year. [3]
- "On average, dog people post a picture or talk about their dog on social media six times per week." [5]

Huge marketing potential exists for lifestyle brands seeking to serve advertisements to demographics associated with certain dog breeds (perhaps owners of active breeds such as golden retrievers, Labradors, etc.) but only a small percentage of actual dog owners are accurately identified. Or suppose you are a social media influence and want to expand your audience base beyond those users that have already expressed interest in certain breeds (via likes and followed accounts). For instance, 157ofGemma is a Spanish-language illustrator (whose Pug is the main subject of her drawings) from Barcelona. How could Instagram suggest US-based Pug fanatics to follow?

From another angle, social media platforms can use this application to classify or expand their existing ad demographics or better understand its user base and social media habits. A platform like Instagram could even make product enhancements to improve the UX for dog owners and celebrity dog accounts.



157ofgemma • Following

157ofgemma Thank you so much Vaiva from @boredpanda for writting about us again ❤️!!!
You can find the link in out stories and in the "INTERVIEW" section on our bio 😊❤️ there is a seleciton of 20 awesome comics, also featuring @thetwinchis!
....
Muchas gracias Vaiva de @boredpanda por escribir sobre nosotros otra vez ❤️!!!
Podéis encontrar el link en nuestro stories y en la sección "INTERVIEWS" en nuestra bio 😊❤️. Hay una selección de 20 de los mejores cómics y también salen @thetwinchis!
#pug #pugcomic #boredpanda #157ofgemma #mochithepug

Load more comments

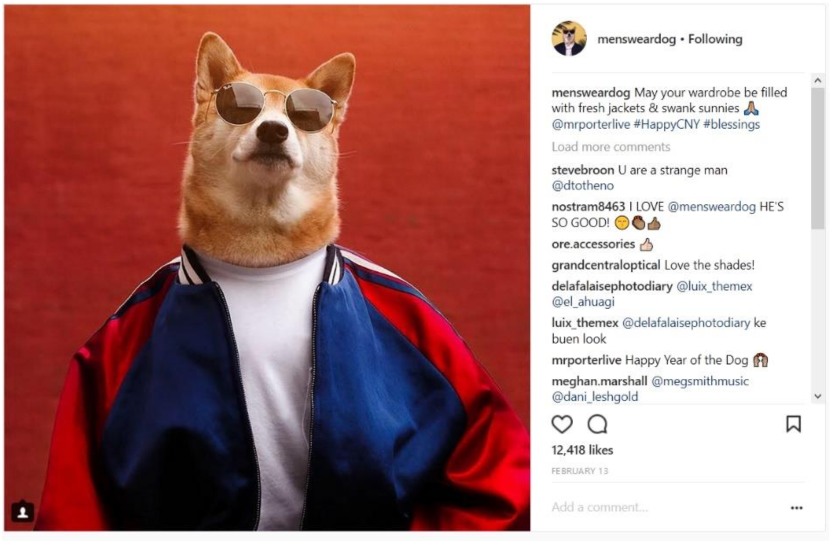
miss.waldorf @antoniocs1988

19,831 likes

6 DAYS AGO

Add a comment...

@157ofgemma- 184K followers. Despite this account being in Spanish and based in Barcelona, much of the content isn't limited by language as she illustrates in both English and Spanish.



mensweardog • Following

mensweardog May your wardrobe be filled with fresh jackets & swank sunnies 🙏 @mrporterlive #HappyCNY #blessings

Load more comments

stevebroon U are a strange man @dtotheno

nostram8463 I LOVE @mensweardog HE'S SO GOOD! 😚👏👍

ore.accessories 👍

grandcentraloptical Love the shades!

delafalaisephotodiary @luix_themex @el_ahuagi

luix_themex @delafalaisephotodiary ke buen look

mrporterlive Happy Year of the Dog 🙌

meghan.marshall @megsmithmusic @dani_leshgold

12,418 likes

FEBRUARY 13

Add a comment...

@Mensweardog – 350K followers

2. Data Acquisition and Overview

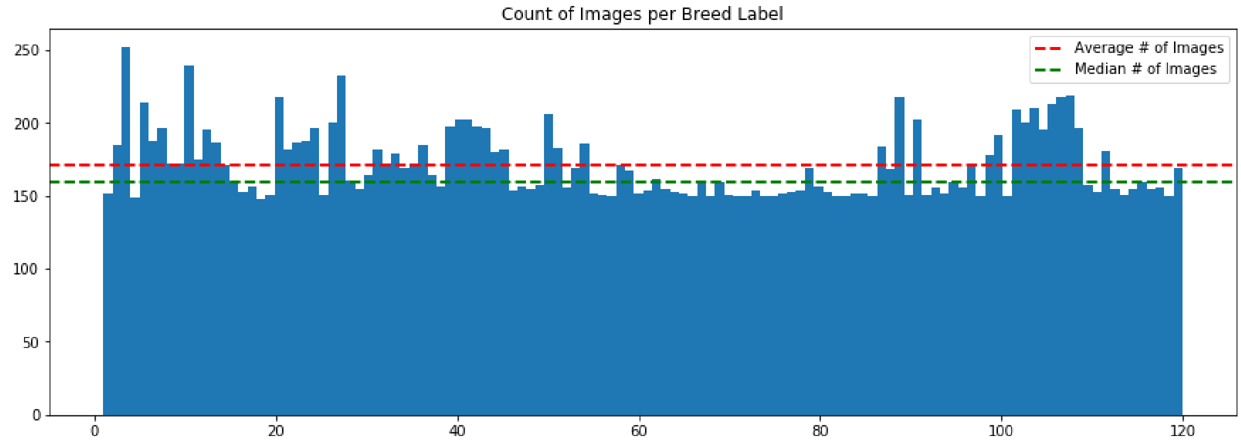
Naturally, the desired dataset would be a collection of pure breed dog images with their associated labels. Luckily academics at Stanford had already attempted this problem and made their acquired dataset publicly available. The dataset of dog images was published by Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Li Fei-Fei at Stanford University (http://vision.stanford.edu/aditya86/ImageNetDogs/).

The full set includes: the raw images which are grouped into folders by their labeled breed, image cropping parameters, and labels. The set also includes a separate file which contains training and testing features (which allows the option to bypass any image conversion and pre-processing). However, I opted to construct the pipeline myself to convert and process the image for training/testing. The images themselves were obtained from ImageNet, a large visual database designed for use in visual object recognition software research.

3. Data Cleaning, Exploration and Analysis

Initially there were a small handful of mis-labeled image data found by previewing some of the images. However, the neural network methods I intend to employ will essentially negate the effects of a small set of mislabeled images. Therefore, there were no attempts to move or re-label any images.

The dataset includes 20,580 labeled images across 120 breeds. The 120 breeds are roughly in line with the 167 that the American Kennel Club currently recognizes (whereas the World Canine Organization recognizes about 340 but would make our dataset unwieldly). The below illustrates the distribution of images per breed:

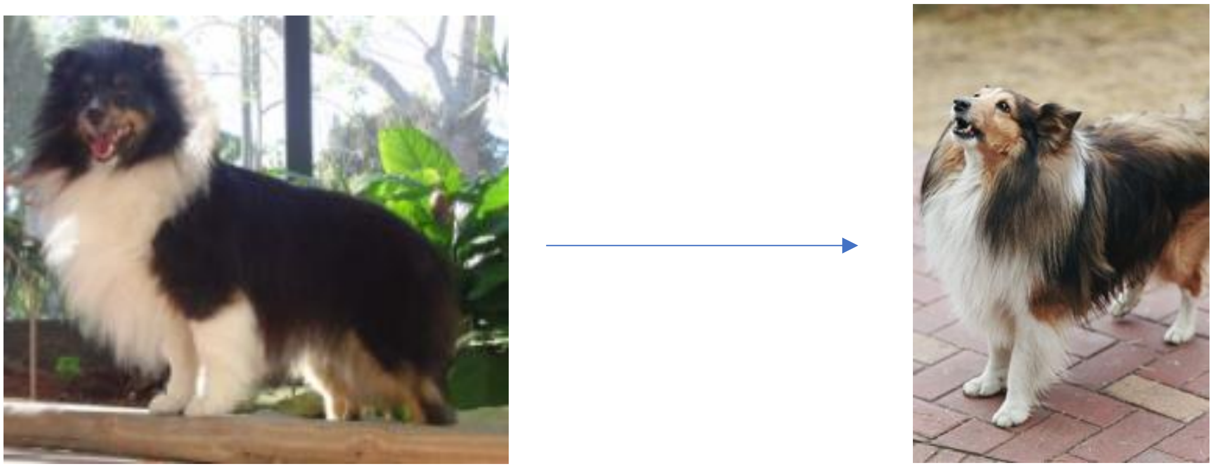


The above bar chart suggests that we have some class imbalance but not to a huge degree. Given that each breed has at least 150 images, the dataset will be stratified such that each class (breed) will have 80 images for training, 20 for validation and anything in excess as test data (on average, about 60 images).

The dataset also includes the images in Numpy arrays and labels and the directory path contained in Matlab database files. Because I opt to use the Keras API (and specifically the flow_from_directory and ImageDataGenerator methods), I will not be using this pre-processed data and using only the images themselves.

The data set includes annotation data which are instructions on how to crop each individual images so that the dog constitutes a higher portion of the image. I took advantage of this meta-data and cropped images after sorting into respective train, validate and test partitions.

Moreover, only one image in the entire dataset was in an incorrect image format. This image on the left had an extra opacity channel (typically associated with .png file extensions) unlike the standard 3 channels (RGB) of .jpg as in the rest of the data set. Since this was the sole outlier, I replaced the image on the left with an image taken from Google Images on the right.



Lastly, loading data into memory for model training became a challenge. As a result, two different methods for processing the image data for training was used. This explains the differences in the coding of how the images are fed into the model. Please see appendix for more details.

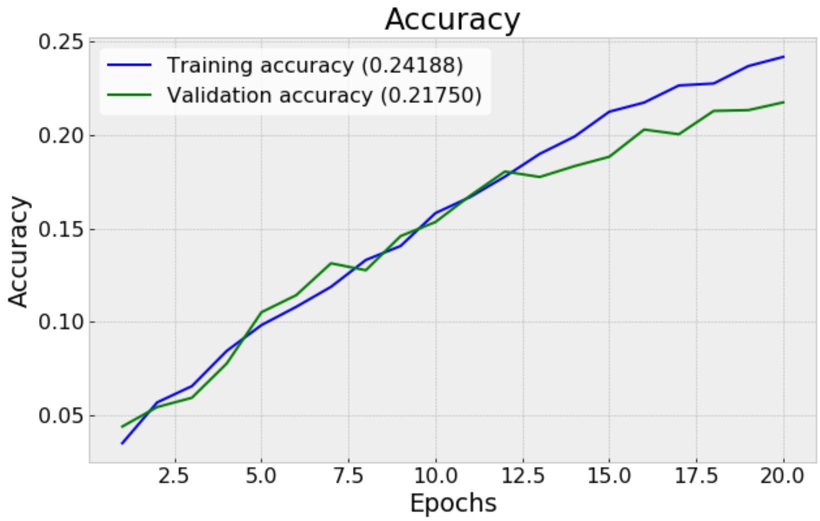
4. Building Convolutional Neural Networks from Scratch

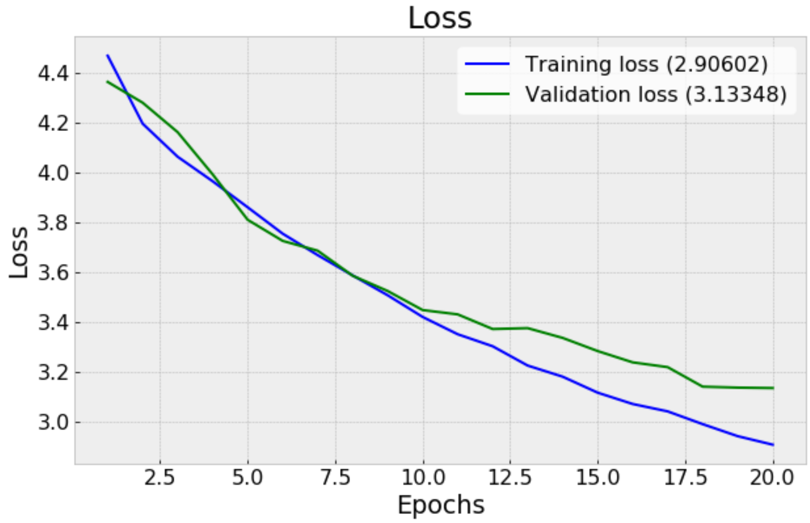
As a benchmark for performance of a pre-trained model and to get a sense of how CNNs work, it was critical to build a CNN network from scratch. The CNN from scratch was inspired by the pre-trained VGG16 model. In this case, the CNN model from scratch became a process of delicately balancing between computational capacity and model accuracy. Through experimentation and several iterations, I found that increasing a CNN's complexity could achieve better accuracy but also could be impossible to train using my own computer.

I experimented with a variety of different architectures using the foundational unit of (Convolutional Layer + MaxPooling + Dropout). Some early iterations included convolutional layers with fewer filters but with large kernel sizes and stride > 1 closer at the input layer and followed by convolutional layers with more filters and decreasing kernel size and stride in the hidden layers near the output. Another iteration included deeper networks which more hidden layers but a consistent kernel size and number of layers. These early iterations failed to achieve an accuracy over 1%.

The final iteration was a CNN model using a BatchNormalization layer as the input layer, followed by 2 layers of a ConvNet layer (64 filters, 3x3, stride of 1), a MaxPooling Layer (2x2) and a BatchNormalization layer. The following 3 layers were similar except for the ConvNet layers had 32 layers instead of 64. The hidden convolutional layers then output to a Dense layer of 2048 nodes which included a Dropout of 20% and GlobalAveragePooling layer (this dramatically decreases the number of trainable parameters but was instrumental to allowing my system to manage the training). The final output layer was 120 nodes (one for each dog breed class) using a SoftMax activation function (the standard for a multi-class application).[1] Finally, Adam optimization function was used with a learning rate of 0.001.

Through experimentation, it became clear that either a slower learning rate would have been too computationally intensive or a faster rate would result in overfitting. Eventually, the optimal learning rate became 0.001. The resulting model includes 426,654 trainable parameters. After 20 initial epochs of training, the CNN from scratch achieved a validation accuracy of 21.7% and a categorical cross-entropy loss of 3.13 as illustrated below:





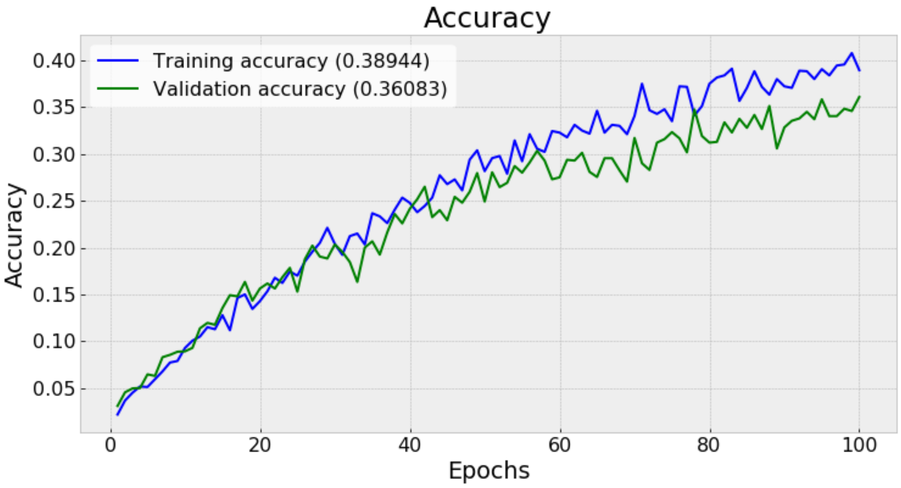
At this stage, I suspected that the loss and accuracy can be improved with more epochs of training. The model was trained for another 20 epochs was but subject to early stopping after just 9. The early stopping parameter was omitted for another 10 epochs to see if training could capture any additional gains. However, the model was only able to attain marginal improvements as shown in the table below:

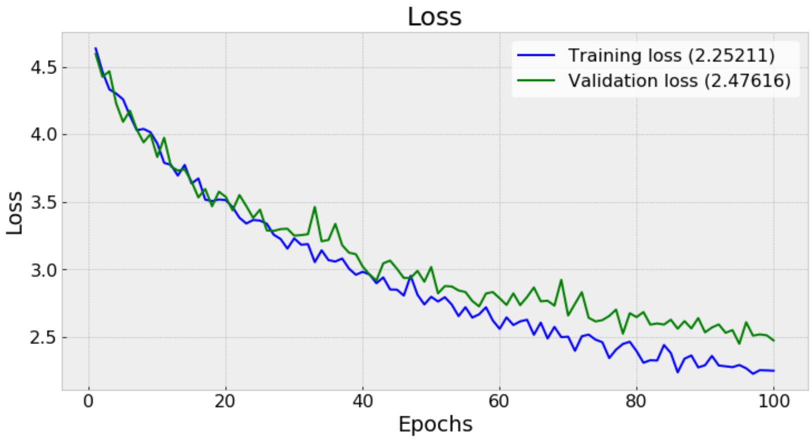
CNN from Scratch #1	After 20 Epochs	...After 29	...After 39
Validation Loss	3.13	3.07 (-0.06)	2.96 (-0.11)
Validation Accuracy	21.7%	23.9% (+2.2%)	25.4% (+1.5%)

Given that validation loss and accuracy were not improving anywhere near the rate as in the initial round of training, it was clear that this model was not going to make any additional significant progress.

It was evident that an overhaul in the architecture was necessary to achieve a higher accuracy rate. Using the VGG16 (Visual Geometry Group) model as inspiration, I constructed a simplified version using a Batch Normalization layer for input, 5 Conv Net layers with increasing number of layers (from 16 to 256), a final hidden layer that included Global Average Pooling, and a dense output layer with 120 nodes with SoftMax Activation. The resulting model has a similar number of trainable parameters of 424,446. The rationale of this architecture is that the Convolutional layers near the input identifies simple lines and edges (which requires fewer number of layers) while the later Convolution layers learns patterns constructed from these lines and edges (with more layers able to distinguish more patterns and complexities). Finally, unlike the first model, the kernel constraint parameter was omitted in favor of Batch Normalization for regularization.

After 100 epochs of training, this updated reached a validation accuracy of 36.1% and a loss of 2.47 as per below:





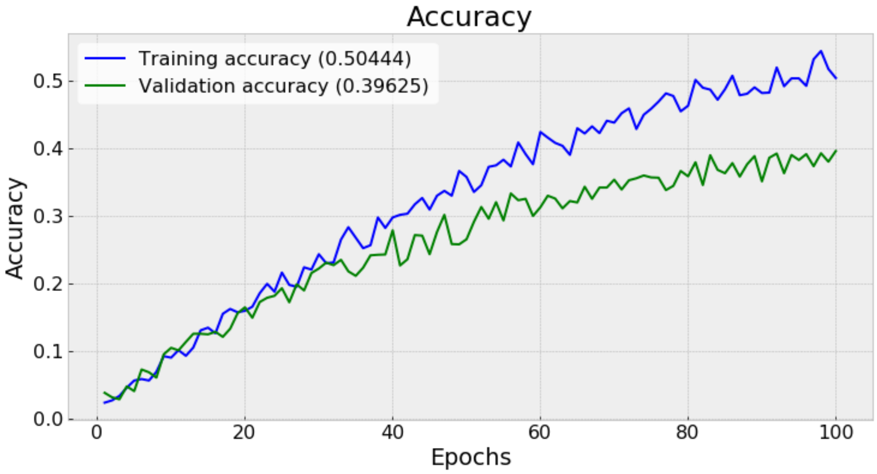
With shifting the number of convolutional layers from the front of the network to the back (closer to output layer), the second version of the CNN achieved a validation accuracy of 36.0% and a loss of 2.47, an improvement over the first version of 11.4% and 0.48, respectively.

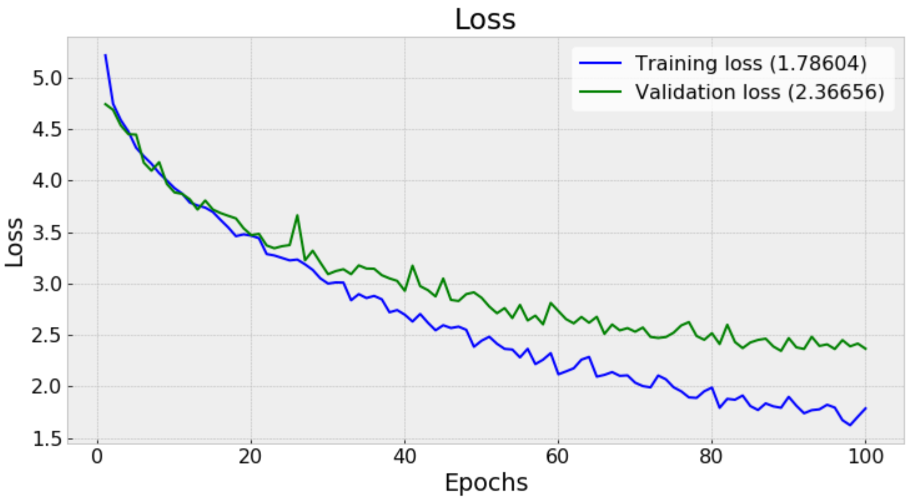
CNN from Scratch #1	After 20 Epochs	…After 29	…After 39	After 100
Validation Loss	3.13	3.07 (-0.06)	2.96 (-0.11)	(n/a)
Validation Accuracy	21.7%	23.9% (+2.2%)	25.4% (+1.5%)	(n/a)
CNN Scratch #2				
Validation Loss	3.5*	3.25 (-0.25)	3.0 (-0.25)	2.48 (-0.52)
Validation Accuracy	15%*	20%* (+5%)	24%* (+4%)	36.0% (+12%)

*Estimated from plot

However, I was not quite satisfied with this iteration of a CNN from scratch as my baseline. I suspected that the plateauing of accuracy and loss combined with diverging accuracy scores between training and validation was due to an overly aggressive dropout rate (40% in this case). I decided to try training a 3rd iteration the same model with a Dropout rate of 20% and one less layer of Batch Normalization before the output layer.

After 100 epochs of training, this 3rd iteration achieved a modest improvement over the prior version:





Instead of curbing overfitting, the changes to the model exacerbate the effect (as the difference between training and validation accuracy widened from 2.9% to 10.4%). However, this version of the model achieved an overall higher validation accuracy of 39.6% (an improvement of 3.6%) and a better loss value of 2.36 (vs. 2.47 previously). The results of training each version is summarized below:

CNN from Scratch #1	After 20 Epochs	…After 29	…After 39	After 100
Validation Loss	3.13	3.07 (-0.06)	2.96 (-0.11)	(n/a)
Validation Accuracy	21.7%	23.9% (+2.2%)	25.4% (+1.5%)	(n/a)
CNN Scratch #2				
Validation Loss	3.5*	3.25* (-0.25)	3.0* (-0.25)	2.48 (-0.52)
Validation Accuracy	15%*	20%* (+5%)	24%* (+4%)	36.0% (+12%)
CNN Scratch #3				
Validation Loss	3.5*	3.3* (-0.2)	3.0* (-0.3)	2.36 (-0.64)
Validation Accuracy	16%*	21%* (+5%)	25%* (+4%)	39.6% (14.6%)

*Estimated from plot

On unseen test data, the third/final CNN from scratch achieved an **accuracy of 40.3%** and a **loss of 2.29**.

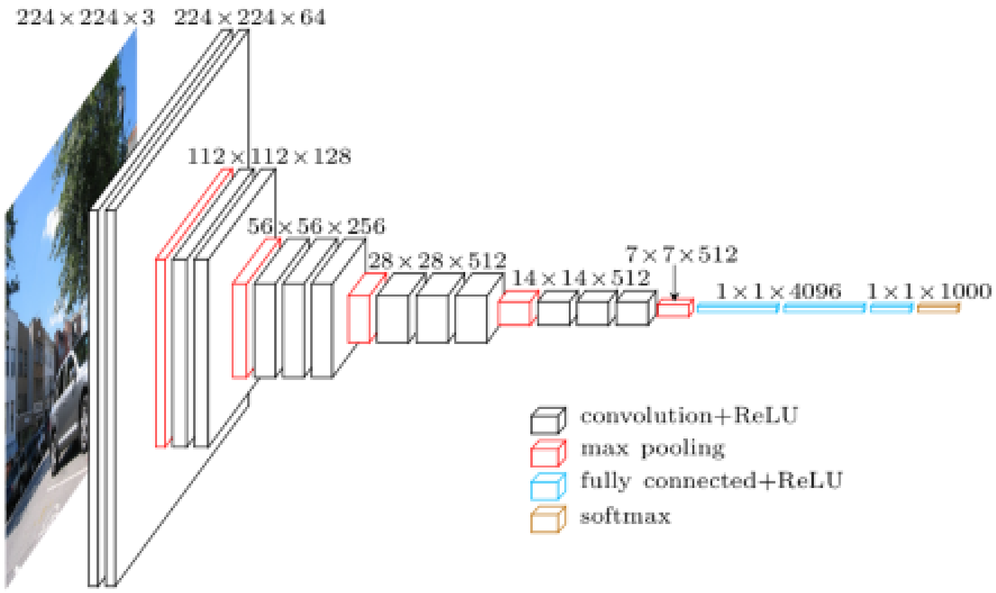
[1] **Explanation of the numerous layers used in the model:**
Batch Normalization: Takes each input and subtracts the batch mean and divides by batch standard deviation. It allows each layer to learn more independently of each layer and helps reduce overfitting.
Convolutional Layer: A set of learnable filters which, during forward pass, convolves across the height and width of an image and computes the dot product and outputs a 2-D activation map.
Max Pooling 2D: Tasked with down sampling the spatial dimensions of the output. In this model, discards around 75% of the activations of the input and keeping only the max values.
Dropout: A regularization method where individual nodes are dropped out of the net , reducing the size of the network . Dropout is implemented to prevent overfitting.
Global Average Pooling 2D: Applies average pooling on the spatial dimensions effectively reducing the input dimensions to 2 dimensions. The resulting output is like Flatten (where Flatten only reshapes input into 2-D).
Dense: A fully-connected layer which performs the classification (in our application, contains 120 nodes for each breed). Each node has a 'SoftMax' activation function which outputs probabilities for each class.

5. Using Transfer Learning (VGG16 with Batch Normalization)

In my research, I found that Convolutional Neural Networks are rarely built and trained from scratch due to the sheer computational power needed. Instead, most applications will utilize a model that has already been trained on a much larger dataset such as the broader ImageNet set (1.2 million images across 1,000 categories). With the pre-trained model, only the subsequent hidden and output layers are trained for the purpose at hand. This is considering using a pre-trained model as a "feature extractor" (more on this later).

I chose the VGG16 model with Batch Normalization using weights from ImageNet because of its notoriety by placing First and Second place (for localization and classification, respectively) for the ImageNet 2014 Challenge[1]. Moreover, the model and has an input shape of 224 x 224 which would result in the least image fidelity loss for the dog image dataset. Furthermore, I opted for the model variant that included Batch Normalization to combat overfitting (given the relatively small data set).

The standard VGG16 network architecture is illustrated as follows:



For our purpose, the final fully connected and output layers were excluded from the original model. Due to the limitations of my computer, I instead swapped in a dropout layer (40%), a fully-connected layer with only 512 nodes (as opposed to 4096 as in the original model), a Batch Normalization and Dropout layer (again 40%), another fully-connected layer of 256 nodes, a dropout layer of 20% and a final output layer of 120 nodes each with SoftMax activation. Despite only these layers being trainable, my total trainable parameter count was 13,009,272 out of a possible 27,725,496.

The below Figure 2 illustrates the differences in model architecture with the Simple CNN from scratch (left) compared with the VGG16 model on the right which boasts a vastly deeper network.

[1] "Very Deep Convolutional Networks for Large-Scale Image Recognition". September 14, 2014. https://arxiv.org/abs/1409.1556

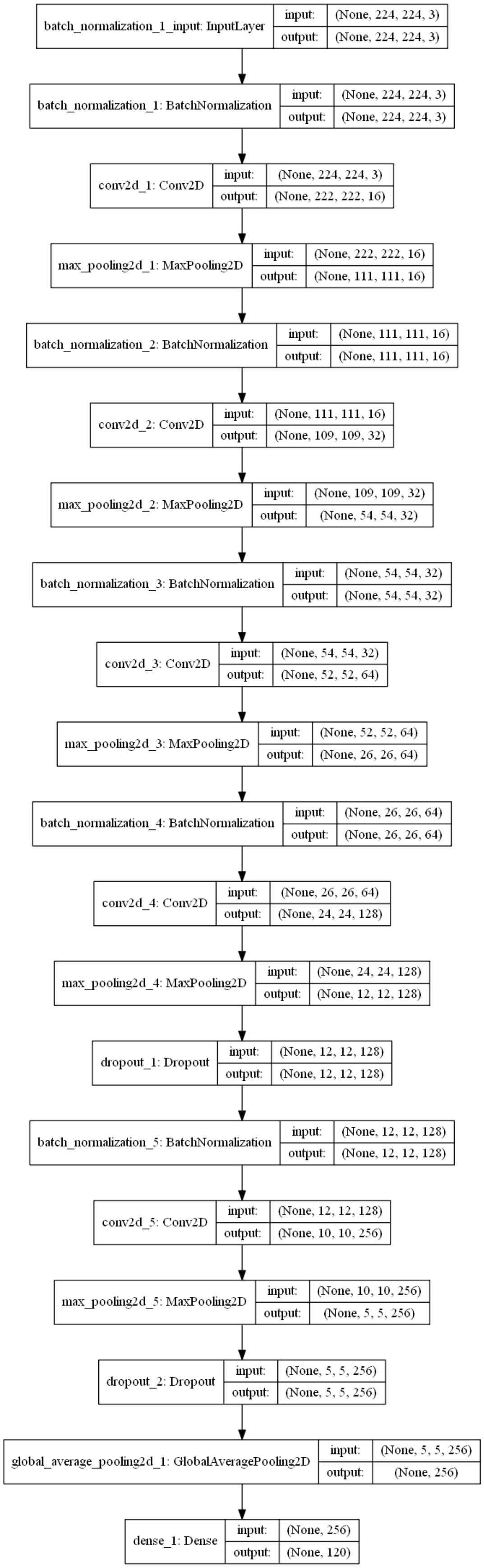
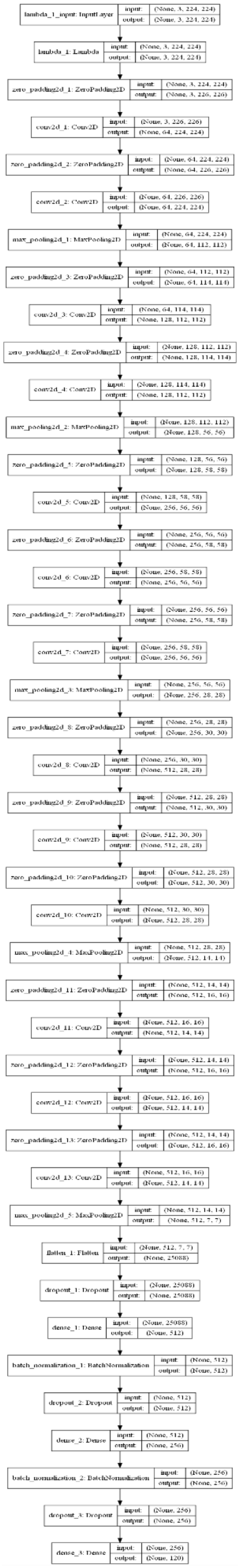
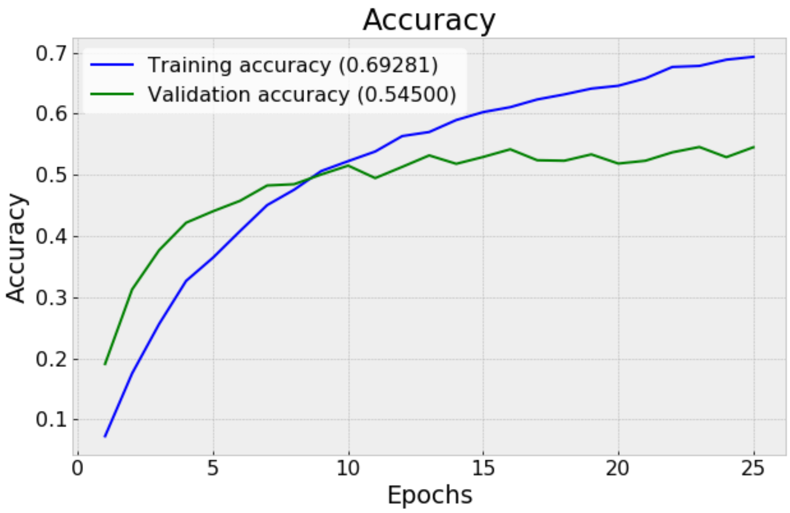
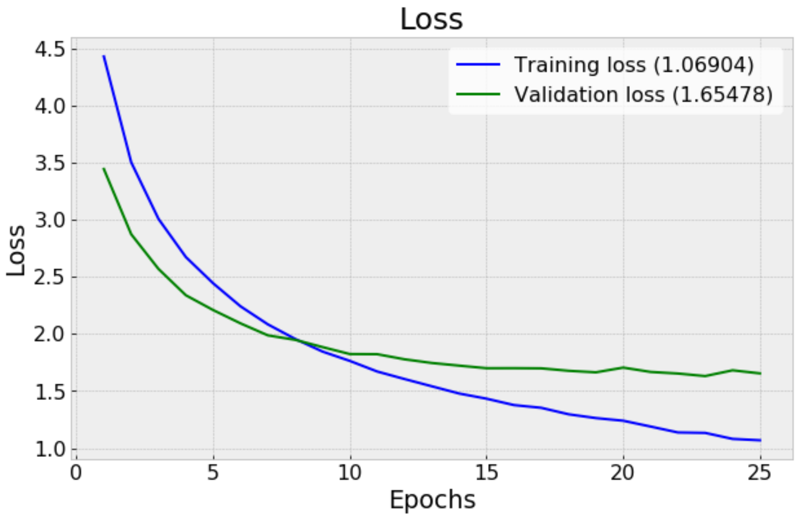

batch_normalization_1_input: InputLayer
input: (None, 224, 224, 3)
output: (None, 224, 224, 3)
batch_normalization_1: BatchNormalization
input: (None, 224, 224, 3)
output: (None, 224, 224, 3)
conv2d_1: Conv2D
input: (None, 224, 224, 3)
output: (None, 222, 222, 16)
max_pooling2d_1: MaxPooling2D
input: (None, 222, 222, 16)
output: (None, 111, 111, 16)
batch_normalization_2: BatchNormalization
input: (None, 111, 111, 16)
output: (None, 111, 111, 16)
conv2d_2: Conv2D
input: (None, 111, 111, 16)
output: (None, 109, 109, 32)
max_pooling2d_2: MaxPooling2D
input: (None, 109, 109, 32)
output: (None, 54, 54, 32)
batch_normalization_3: BatchNormalization
input: (None, 54, 54, 32)
output: (None, 54, 54, 32)
conv2d_3: Conv2D
input: (None, 54, 54, 32)
output: (None, 52, 52, 64)
max_pooling2d_3: MaxPooling2D
input: (None, 52, 52, 64)
output: (None, 26, 26, 64)
batch_normalization_4: BatchNormalization
input: (None, 26, 26, 64)
output: (None, 26, 26, 64)
conv2d_4: Conv2D
input: (None, 26, 26, 64)
output: (None, 24, 24, 128)
max_pooling2d_4: MaxPooling2D
input: (None, 24, 24, 128)
output: (None, 12, 12, 128)
dropout_1: Dropout
input: (None, 12, 12, 128)
output: (None, 12, 12, 128)
batch_normalization_5: BatchNormalization
input: (None, 12, 12, 128)
output: (None, 12, 12, 128)
conv2d_5: Conv2D
input: (None, 12, 12, 128)
output: (None, 10, 10, 256)
max_pooling2d_5: MaxPooling2D
input: (None, 10, 10, 256)
output: (None, 5, 5, 256)
dropout_2: Dropout
input: (None, 5, 5, 256)
output: (None, 5, 5, 256)
global_average_pooling2d_1: GlobalAveragePooling2D
input: (None, 5, 5, 256)
output: (None, 256)
dense_1: Dense
input: (None, 256)
output: (None, 120)

Figure 2 – Visual map of the architecture contained within the CNN model from Scratch (left) vs. the VGG16 model (right). Note the degree of depth of VGG16 (16 hidden layers) compared with the Simple model.


lambda_1_input: InputLayer
input: (None, 3, 224, 224)
output: (None, 3, 224, 224)
lambda_1: Lambda
input: (None, 3, 224, 224)
output: (None, 3, 224, 224)
zero_padding2d_1: ZeroPadding2D
input: (None, 3, 224, 224)
output: (None, 3, 226, 226)
conv2d_1: Conv2D
input: (None, 3, 226, 226)
output: (None, 64, 224, 224)
zero_padding2d_2: ZeroPadding2D
input: (None, 64, 224, 224)
output: (None, 64, 226, 226)
conv2d_2: Conv2D
input: (None, 64, 226, 226)
output: (None, 64, 224, 224)
max_pooling2d_1: MaxPooling2D
input: (None, 64, 224, 224)
output: (None, 64, 112, 112)
zero_padding2d_3: ZeroPadding2D
input: (None, 64, 112, 112)
output: (None, 64, 114, 114)
conv2d_3: Conv2D
input: (None, 64, 114, 114)
output: (None, 128, 112, 112)
zero_padding2d_4: ZeroPadding2D
input: (None, 128, 112, 112)
output: (None, 128, 114, 114)
conv2d_4: Conv2D
input: (None, 128, 114, 114)
output: (None, 128, 112, 112)
max_pooling2d_2: MaxPooling2D
input: (None, 128, 112, 112)
output: (None, 128, 56, 56)
zero_padding2d_5: ZeroPadding2D
input: (None, 128, 56, 56)
output: (None, 128, 58, 58)
conv2d_5: Conv2D
input: (None, 128, 58, 58)
output: (None, 256, 56, 56)
zero_padding2d_6: ZeroPadding2D
input: (None, 256, 56, 56)
output: (None, 256, 58, 58)
conv2d_6: Conv2D
input: (None, 256, 58, 58)
output: (None, 256, 56, 56)
zero_padding2d_7: ZeroPadding2D
input: (None, 256, 56, 56)
output: (None, 256, 58, 58)
conv2d_7: Conv2D
input: (None, 256, 58, 58)
output: (None, 256, 56, 56)
max_pooling2d_3: MaxPooling2D
input: (None, 256, 56, 56)
output: (None, 256, 28, 28)
zero_padding2d_8: ZeroPadding2D
input: (None, 256, 28, 28)
output: (None, 256, 30, 30)
conv2d_8: Conv2D
input: (None, 256, 30, 30)
output: (None, 512, 28, 28)
zero_padding2d_9: ZeroPadding2D
input: (None, 512, 28, 28)
output: (None, 512, 30, 30)
conv2d_9: Conv2D
input: (None, 512, 30, 30)
output: (None, 512, 28, 28)
zero_padding2d_10: ZeroPadding2D
input: (None, 512, 28, 28)
output: (None, 512, 30, 30)
conv2d_10: Conv2D
input: (None, 512, 30, 30)
output: (None, 512, 28, 28)
max_pooling2d_4: MaxPooling2D
input: (None, 512, 28, 28)
output: (None, 512, 14, 14)
zero_padding2d_11: ZeroPadding2D
input: (None, 512, 14, 14)
output: (None, 512, 16, 16)
conv2d_11: Conv2D
input: (None, 512, 16, 16)
output: (None, 512, 14, 14)
zero_padding2d_12: ZeroPadding2D
input: (None, 512, 14, 14)
output: (None, 512, 16, 16)
conv2d_12: Conv2D
input: (None, 512, 16, 16)
output: (None, 512, 14, 14)
zero_padding2d_13: ZeroPadding2D
input: (None, 512, 14, 14)
output: (None, 512, 16, 16)
conv2d_13: Conv2D
input: (None, 512, 16, 16)
output: (None, 512, 14, 14)
max_pooling2d_5: MaxPooling2D
input: (None, 512, 14, 14)
output: (None, 512, 7, 7)
flatten_1: Flatten
input: (None, 512, 7, 7)
output: (None, 25088)
dropout_1: Dropout
input: (None, 25088)
output: (None, 25088)
dense_1: Dense
input: (None, 25088)
output: (None, 512)
batch_normalization_1: BatchNormalization
input: (None, 512)
output: (None, 512)
dropout_2: Dropout
input: (None, 512)
output: (None, 512)
dense_2: Dense
input: (None, 512)
output: (None, 256)
batch_normalization_2: BatchNormalization
input: (None, 256)
output: (None, 256)
dropout_3: Dropout
input: (None, 256)
output: (None, 256)
dense_3: Dense
input: (None, 256)
output: (None, 120)

Training this model over 25 epochs for around 3 hours yielded the following:





With a pre-trained model, I was able to achieve a 14.9% improvement on validation accuracy, from 39.6% to 54.5% (in fewer training epochs though still across an equal amount of training time). Loss also improved by 0.71 to 1.65 from 2.36. I wanted to see if these values could improve with more training and a slower learning rate (to combat the overfitting trend).

Pre-Trained VGG16	After 25 Epochs	…After 40 epochs		
Validation Loss	1.65	1.70 (+0.05)		
Validation Accuracy	54.5%	53.1% (-1.4%)		
CNN Scratch #3	After 20 Epochs	…After 29	…After 39	After 100
Validation Loss	3.5*	3.3* (-0.2)	3.0* (-0.3)	2.36 (-0.64)
Validation Accuracy	16%*	21%* (+5%)	25%* (+4%)	39.6% (14.6%)

We find that neither validation loss nor accuracy saw improvements, but rather found some performance degradation. The model was shown to be improving only on the training data but this may be detrimental in the models ability to generalize to new/unseen data.

Overall, the pre-trained model with additional fine-tuning for the application of dog breed classification seemed to be a substantial improvement over the CNN built from scratch with no pre-determined weights. Now I wanted to evaluate the model on test data and determine how well the network performs on new images and if we can infer ways to further improve the network.

6. Evaluating the Pre-Trained Convolutional Neural Network

6a. On test data

After passing in the test data, the model achieved an **accuracy of 58.9%** and a categorical cross-entropy **loss of 1.41**. I found this is rather impressive as I do not believe I could reach the same level of accuracy across 120 dog breeds. Let's dive deeper into the inner workings of the network and gain intuition on what the network is getting wrong. For example, is there a specific pair of breeds the network is having difficulty with? Is there a common trait in the breeds it has trouble with? Is the model predicting a label based on the environmental objects in a photo (such as associating water or snow with certain breeds)?

The confusion matrix in Figure 2 suggests that recall and precision is relatively consistent across the breeds. The matrix plots the frequency of instances where images belonging to a certain class were classified against itself and all others. Lighter color boxes indicate higher frequency of instances. Our ideal outcome is a single unbroken line going down horizontally (indicating 100% accuracy and no additional misclassifications). Instead, the confusion matrix suggests that our model is not as accurate as it can be. Furthermore, we do see a few small clumps of white boxes which indicates a higher frequency of misclassification to related peers. This makes intuitive sense as the image folders are organized such that breeds are grouped together (that is, all the terrier breeds are next to the other terrier breeds) and likely have fewer class variations compared to other breeds.

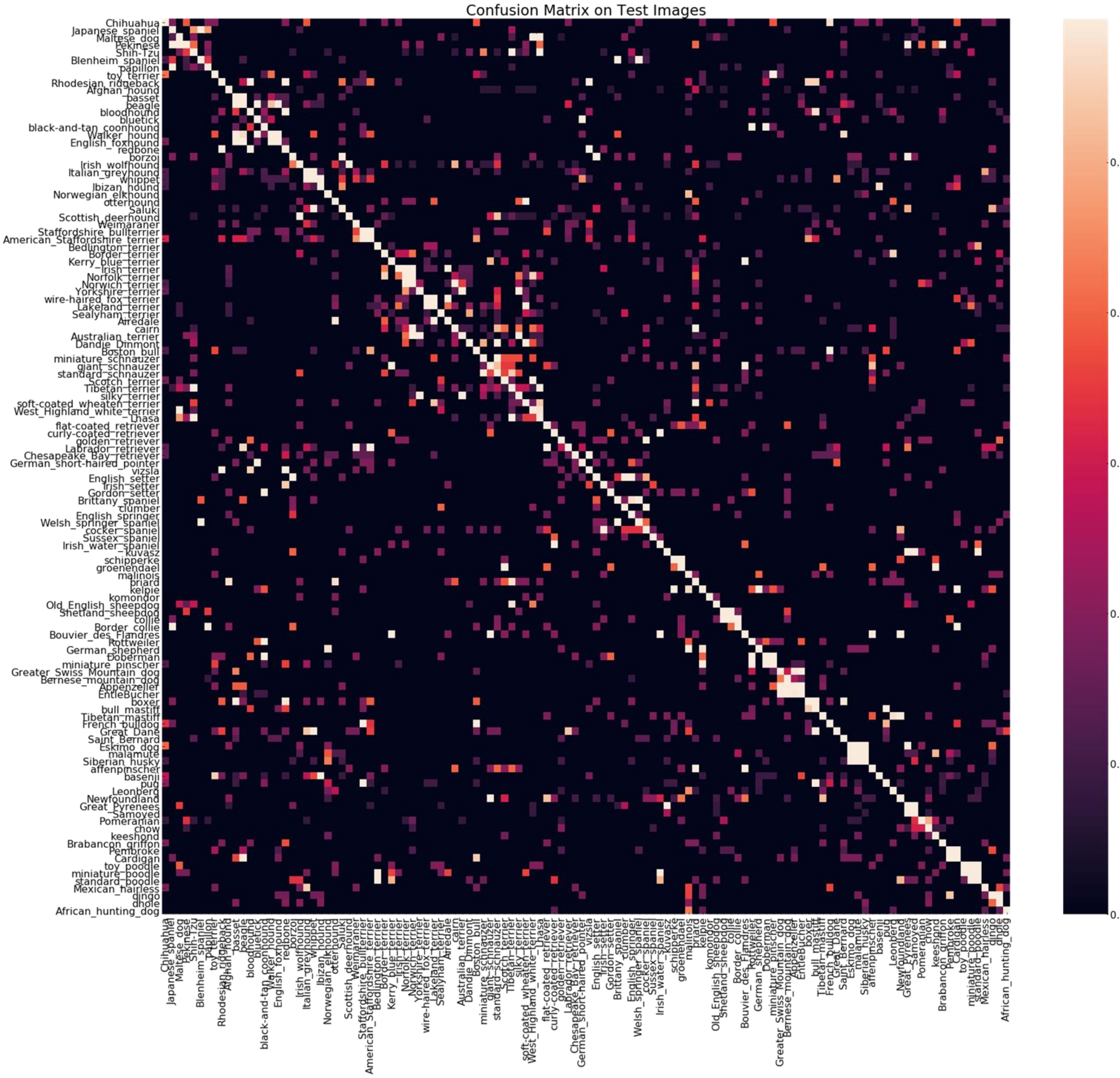


Fig. 2 – Confusion Matrix Heatmap on Test Image Data

However, it is rather difficult to discern patterns at the breed level. Instead it may be more effective to view the top most misclassified breed pairs (Fig. 2):

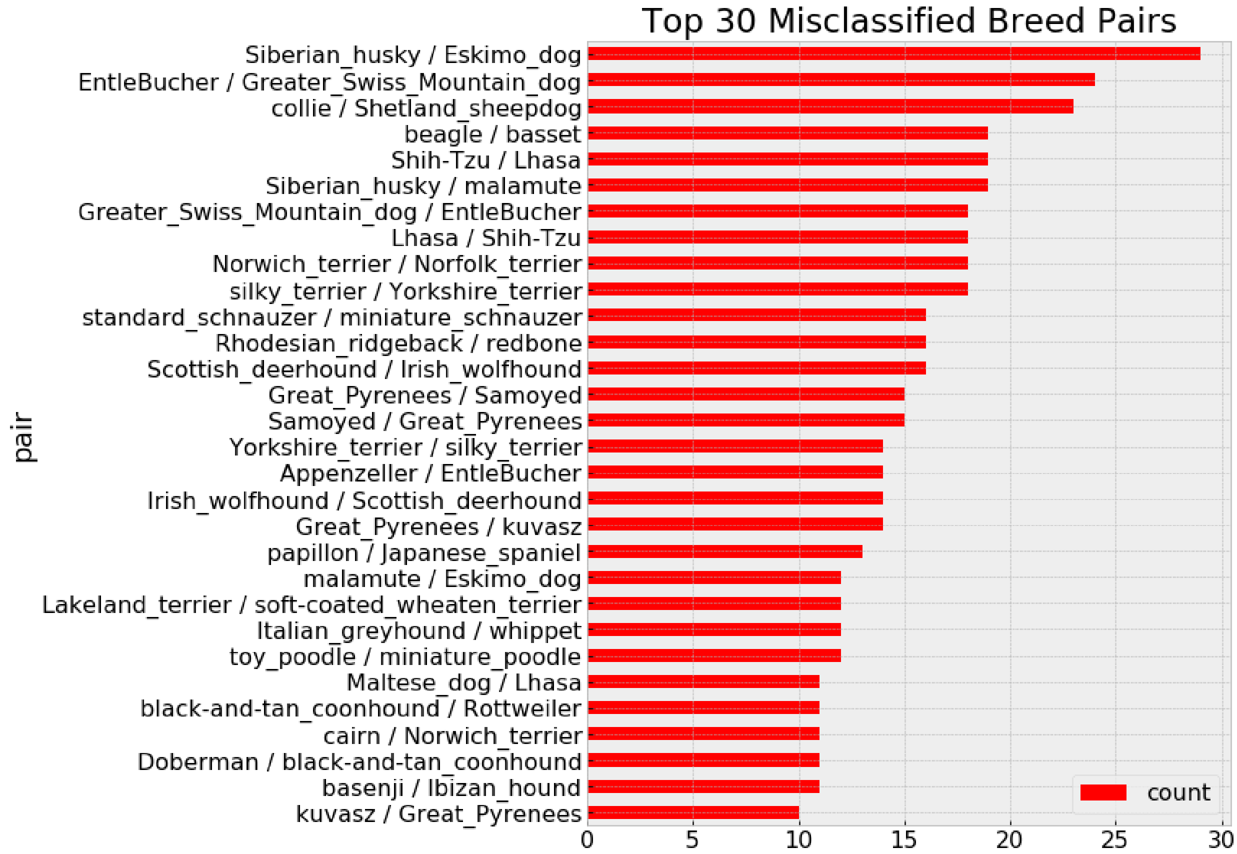
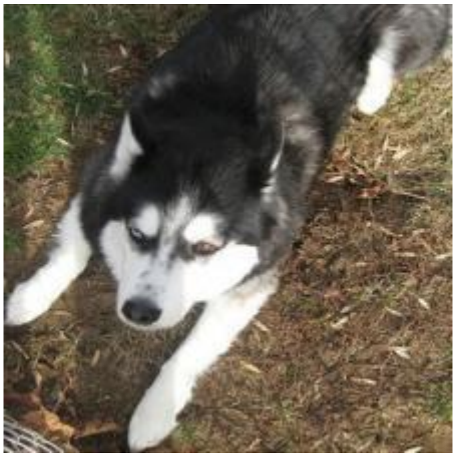
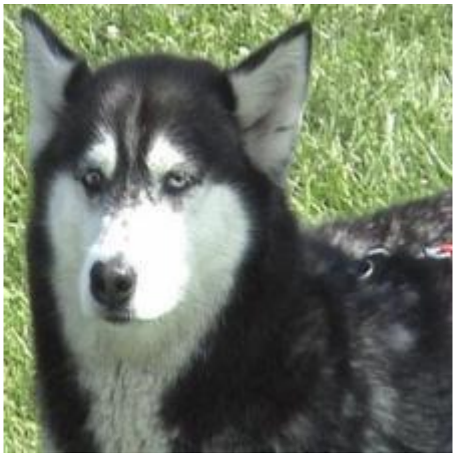


Fig. 2 – Top 30 Most Misclassified Breed Pairs (descending order)

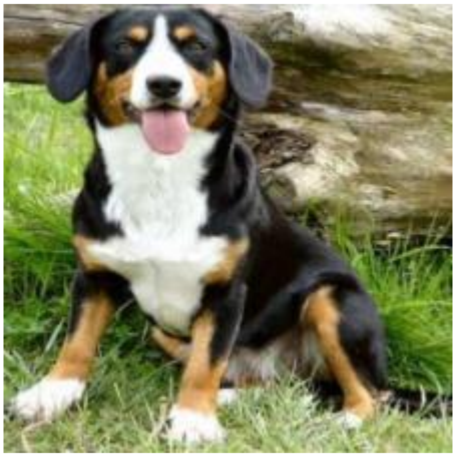
According to the above plot, the Siberian Husky and Eskimo dogs are the most misclassified breeds. This pair misclassification occurred in 30 images combined between the two breeds (Siberian Husky test images: 92, Eskimo Dog test images: 50). This amounts to just a ~21% misclassification rate, not bad considering this is the most confused pair. Image examples of the top misclassified breeds:

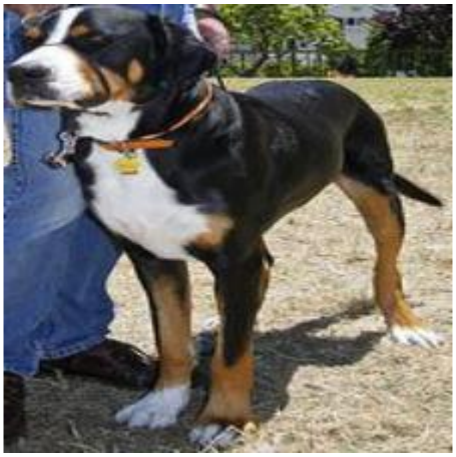
Siberian Husky (left) vs. Eskimo Dog (right)



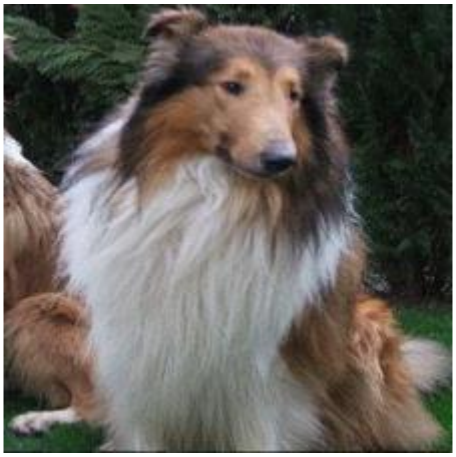


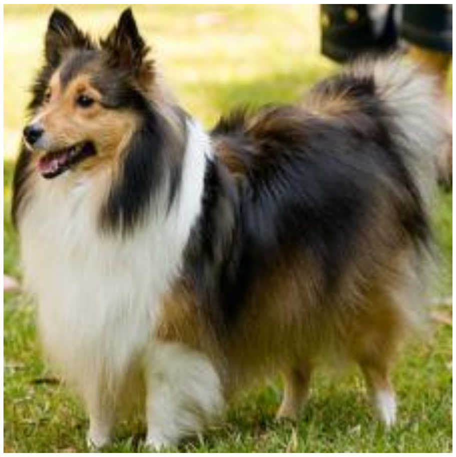
Entlebucher (left) vs. Greater Swiss Mountain Dog (right)





Collie (left) vs. Shetland Sheepdog (right)





From these images of the 3 most commonly misclassified breeds, it is understandable the model struggles to discern the difference and would likely be difficult at the average human-level. Based on these examples, it is difficult to judge whether more training data or additional model training would result in being able to improve accuracy between these breeds.

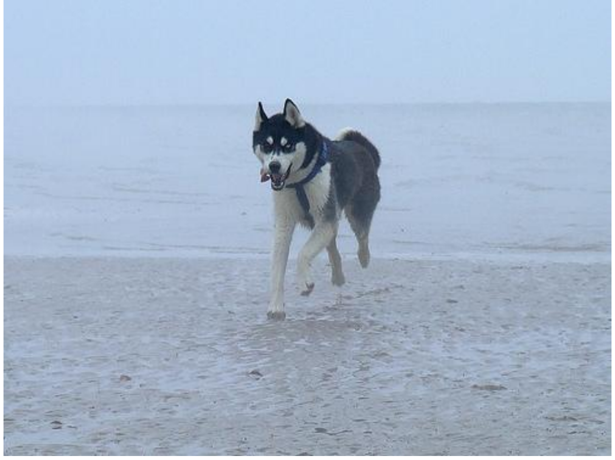
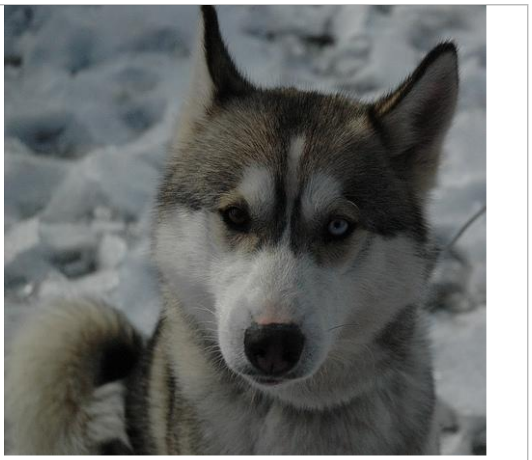
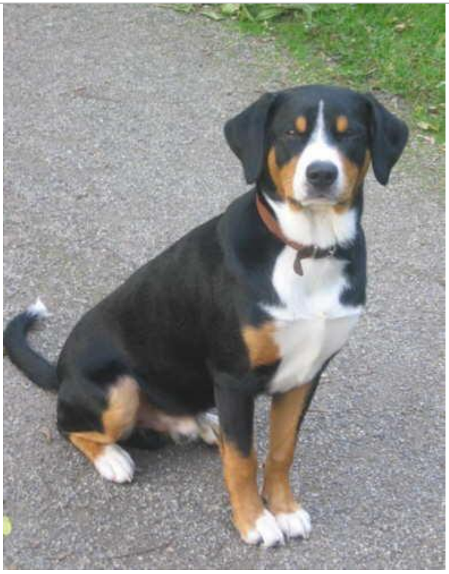
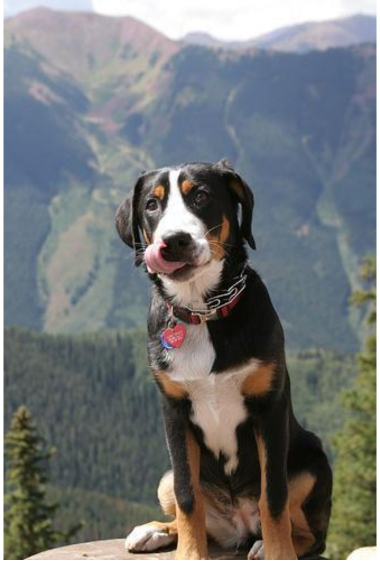
Sci-Kit Learn also provides a Classification Report function that provides additional insight on how the model performs at the class level. In the following table are the highest and lowest values for Precision, Recall and F1-Scores.

Classification Metric	Lowest	Highest	Average
Precision *(TP/TP+FP)*	 Briard – 31%	 Komondor – 89%	60%
Recall *(TP/FN + TP)*	 Collie – 21%	 Norwegian Elkhound – 91%	59%
F1-Score ***(2) / (1/Precision + 1/Recall)***	 Collie – 28%	 Komondor – 88%	59%

It seems the model is generally more accurate on breeds with very distinct physical features or share the fewest similarities with other breeds. I suspect this to also be the case at the human level.

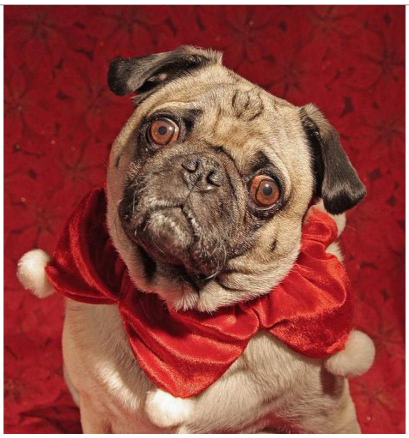
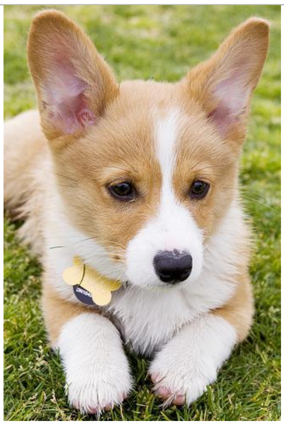
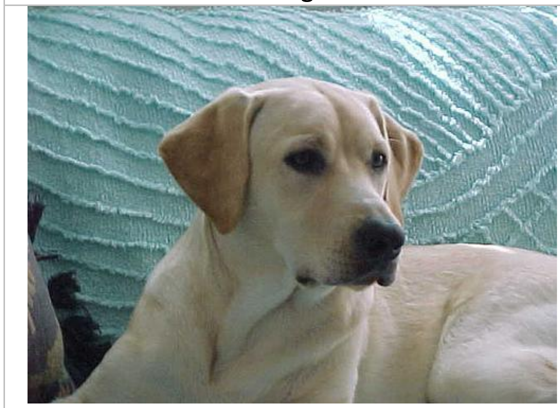
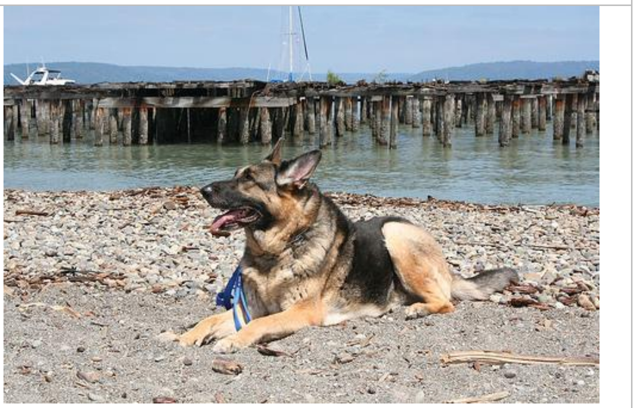
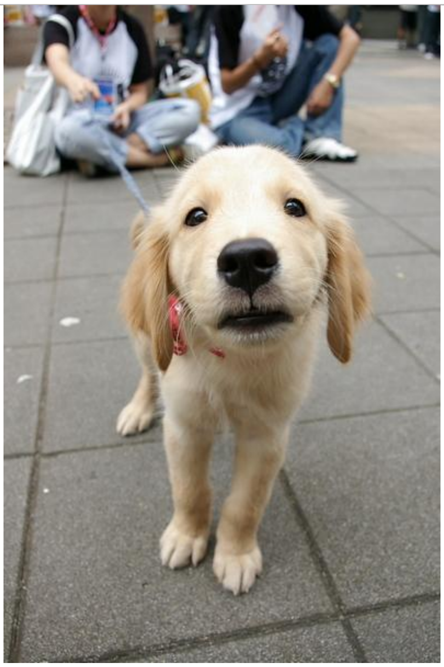
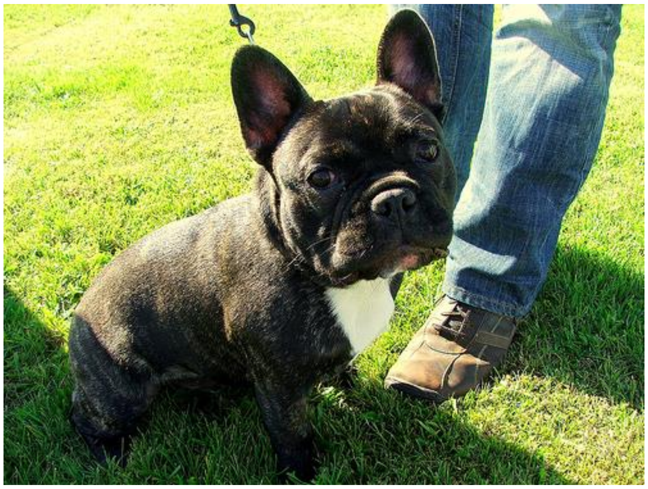
Passing the test images into the model outputs the probability of each image belonging to a certain breed across all possibly breeds. Using this probability data to my advantage, I aggregated the predictions to find the most confident predictions per breed and the respective image. The objective is to visualize images that were maximally representative of each breed as perceived by the model. The following are a selection of such images:

First Set: Top Misclassified Breed Pairs

 Siberian Husky	 Eskimo Dog**
 EntleBucher	 Greater Swiss Mountain Dog

**Interestingly, the model's concept of a maximally representative Eskimo dog is a picture from the Siberian Husky image set. Unfortunately, I cannot discern if the model is correctly labelling the image and the image itself is mis-labeled or if the model is in fact confusing between the two breeds.

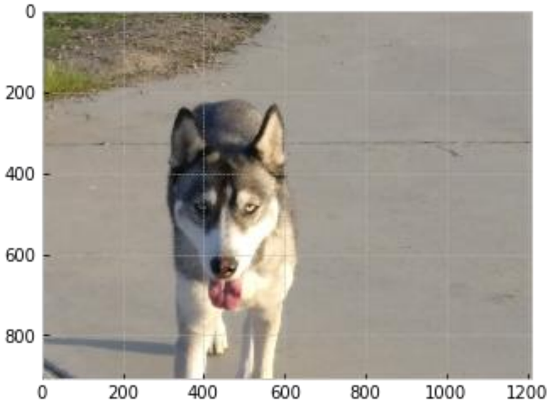
Second Set: Popular Dog Breeds (in U.S.)

 Pug	 Pembroke Welsh Corgi
 Labrador Retriever	 German Shepherd
 Golden Retriever	 French Bulldog

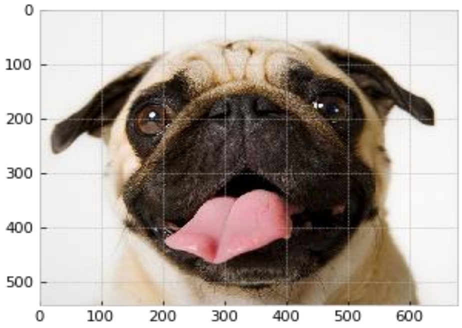
The above images suggest that the pre-trained model has a good grasp of what some breeds are expected to look like. Furthermore, the model seems to be properly weighing the physical characteristics of the dog in the image itself for its classification and not the environmental features (such as grass or water) or appearance of other objects (such as humans or toys).

6b. Testing the model on user derived images

To evaluate the model on an ad-hoc basis, I built a simple function which takes in images from a hard drive directory and outputs a prediction. Let's use the image I took to help locate the lost dog from my story followed by some other ad-hoc examples:

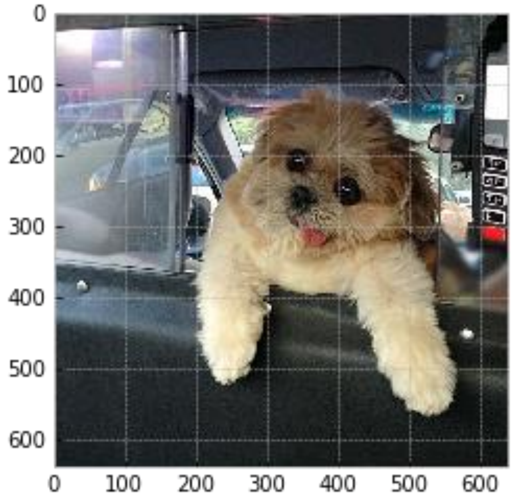
Uploaded Image... Woof! The model predicted the breed as...malinois! ...with a confidence of 40.51%.  0 200 400 600 800 0 200 400 600 800 1000 1200 This appears to be incorrect…. Or perhaps I was wrong in assuming it was a husky?	Malinois I was likely correct about the lost dog being a husky.

Real World Test #2: Pug

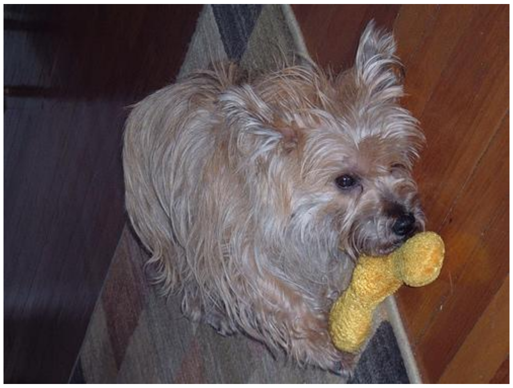
Uploaded Image... Woof! The model predicted the breed as...pug! ...with a confidence of 48.72%.  0 100 200 300 400 500 0 100 200 300 400 500 600	Correct!

Real World Test #3: Shih Tzu (@marniethedog)

```
Uploaded Image...
Woof! The model predicted the breed as...cairn!
...with a confidence of 35.88%.
```

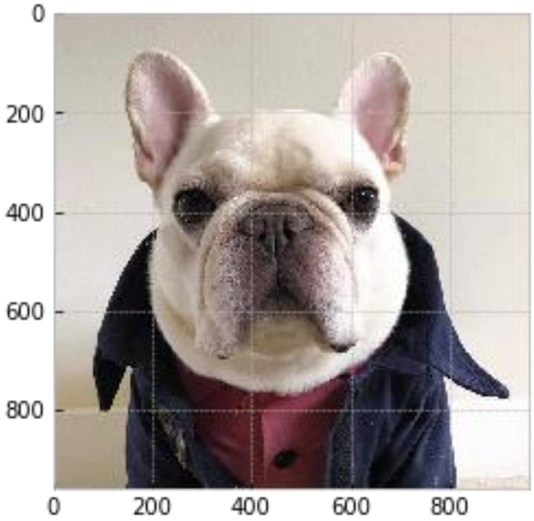


Incorrect: Marnie is a Shih Tzu. Cairn below for reference



Real World Test #4: French Bulldog (@barkleysircharles)

```
Uploaded Image...
Woof! The model predicted the breed
as...French_bulldog!
...with a confidence of 97.15%.
```



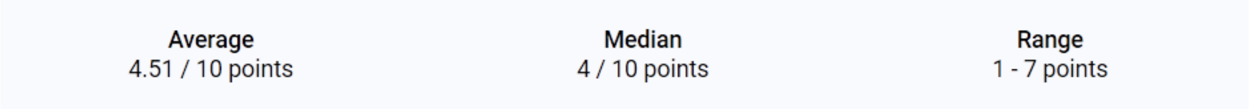
Correct!

It seems the model performs the best on pictures where the dog's face takes up a larger proportion of the image.

6c. How does the model perform against humans?

To test how well the pre-trained model can perform against humans, I devised an informal multiple-choice quiz using Google Forms. I constructed the quiz to have 10 images from the test set with multiple choice selections. For the 10 images, I deliberately choose 5 breeds that the model commonly misclassifies and 5 based on random choice (using Numpy's Random module). For each of the 10 images, I provided: 1 correct breed selection, 2 options based on the most common misclassification from the confusion matrix, and 2 on random choice.

Soliciting an open invitation to my friends on Facebook and LinkedIn, I received 51 respondents (majority from Facebook). Of the identified respondents, 70% were female and 30% were male. The entire sample are between the ages of 21 and 35. 12% of responders chose to remain anonymous and thus no demographic data exists. The results followed a normal distribution with a mean score of 45.1% or 4.51 correct out of 10. The highest score was 7 of 10 correct selections with only 3 respondents (or 5.88% of sample) reaching this level of accuracy. Figure 3 plots the distribution of points while the table in Figure 4 examines the responses at the question level along with the model's predictions.



Average	Median	Range
4.51 / 10 points	4 / 10 points	1 - 7 points

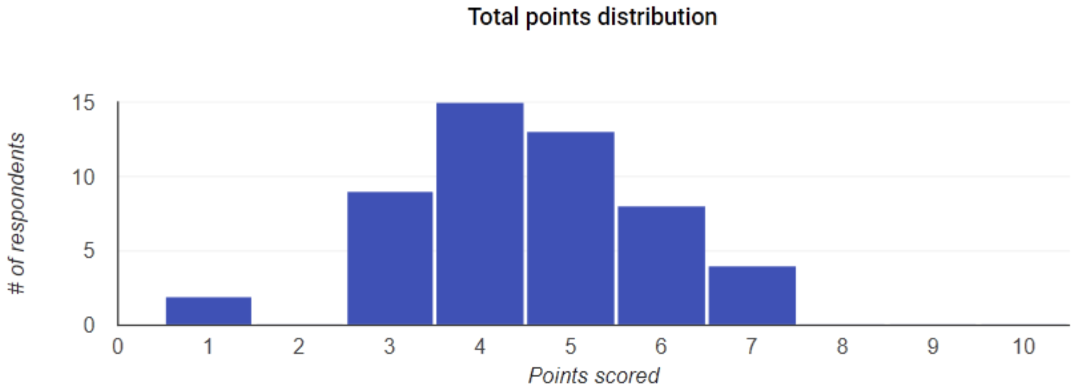


Figure 3 – Point Distribution of n=51

By comparison, the pre-trained model scored 8 out of 10 (or 80% accuracy on the quiz images). I found that humans commonly misclassified similar breeds that the model struggled with. For example, 51% guessed the Collie was a Shetland Sheepdog, 90.2% incorrectly guessed the Eskimo Dog was a husky (the model got this one wrong too) and 74.5% incorrectly guessed the Cairn was a Norwich Terrier.

Correct Label	Human Prediction*	Model Prediction**
Papillion	Papillion (70.0%)	Papillion (99.50%)
Eskimo Dog	Husky (90.0%)	Saluki (18.06%)
EntleBucher	Greater Swiss Mountain Dog (40.0%)	EntleBucher (86.99%)
Cairn	Norwich Terrier (76.0%)	West Highland Terrier (54.76%)
Malamute	Malamute (46.9%)	Malamute (51.96%)
Gordon Setter	Brittany Spaniel (62.0%)	Gordon Setter (97.43%)
Border Collie	Shetland Sheepdog (50.0%)	Border Collie (45.00%)
Shih-Tzu	Shih Tzu (84.0%)	Shih Tzu (15.16%)
French Bulldog	French Bulldog (80.0%)	French Bulldog (97.15%)
Tibetan Mastiff	Tibetan Mastiff (70.0%)	Tibetan Mastiff (85.87%)
(Overall Score)	**Collective: 50% (Average: 44.9%)**	**80.0%**

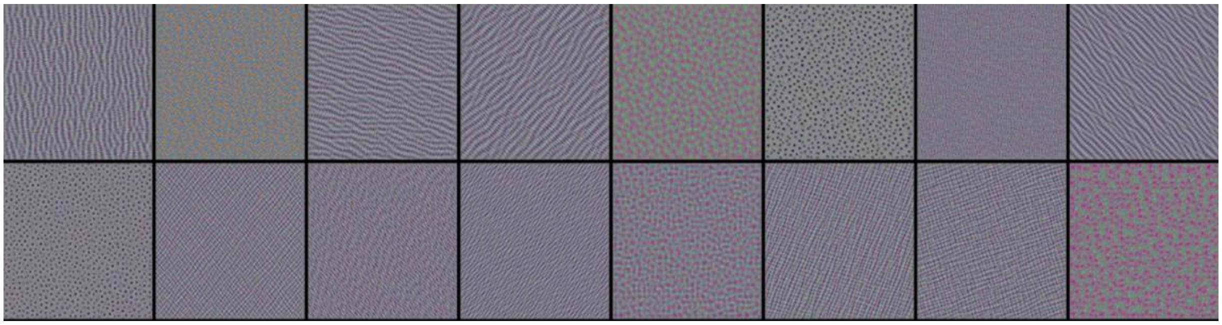
* Majority Response (% of all responses)

** Model Prediction with Confidence

Figure 4 – Comparison of Human Prediction vs. Model Predictions and Scores

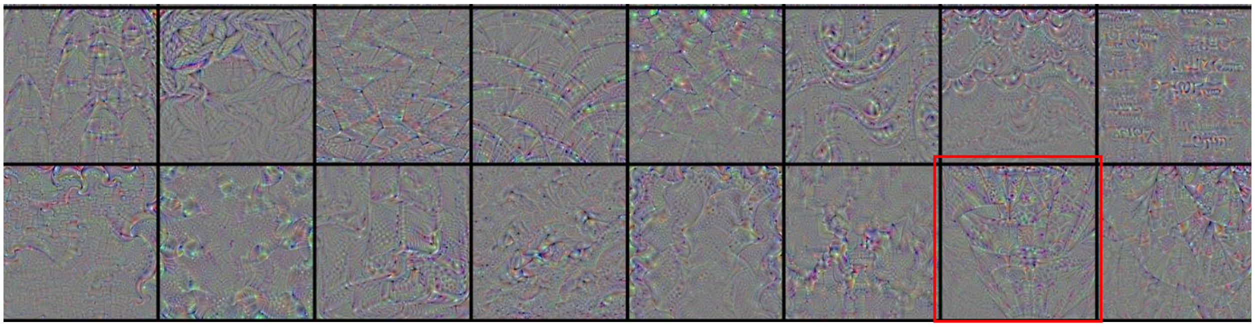
6d. Seeing what the network sees

In examining the network, I extracted 64 visualizations of the most weighted convolutional layers and what they are filtering for. In the below grid, we get a sense of the types of patterns the hidden layers are looking for. In this instance, this is what the 5th convolutional layer is seeing:



These are visual representations of the "features" from the pre-trained VGG16 model. We see that early in the network, the convolutional layers are looking for relatively simple uniform patterns such as dots and lines. These patterns may be a combination of even lower-level textures and patterns from training on the ImageNet data set. At this hidden layer of the network, it does not seem the network is identifying specific shapes or more complex patterns that we would typically associate with a dog or its breed.

In comparison, the 14th convolutional layer appears to be looking for more complex features including larger shapes and silhouettes. Again, this network was trained on a broader image data set and it is possible to begin to make out patterns that resemble everyday objects like flowers or automobiles. For example, in the red highlighted filter in the 4th row and 2nd from right has a silhouette resembling a bouquet.



Despite that this hidden layer is optimized for detecting a plethora of physical objects (other than dogs and their breed), the pre-trained model still achieves a high accuracy relative to the CNN from scratch.

7. Technical Limitations and Future Work/Research

7a. Processing/Computational Limitations

I suspect to have completed this project in a shorter amount of time with the help of cloud services and on-demand high-performance computing (such as Amazon Web Services). The ability to leverage GPUs and processing power in the cloud would have led to faster model iteration and training and resulted in better accuracy and loss metrics.

In my experience on this project, I found that the large demand for onboard memory and a dated GPU was an obstacle to proper training and had to settle for long training times and lower accuracy for the sake of avoiding resource exhaust. The onboard memory challenge became a moot point after I was able to use the Flow_from_directory method from Keras when training the pre-trained VGG16 model. This method iterates over a user-defined batch size over a directory and assumes class labels from the sub-directory names and is less demanding of RAM.

7b. Data Limitations

Despite a large overall quantity of images, at only 80 images per breed dedicated to training was insufficient for a CNN to reach its full learning potential and be able to generalize to new images. If I had more time, I would have spent another few days gathering more labelled images for each breed. I believe that doubling the quantity of training images would be sufficient for significant gains in model accuracy.

7c. Future Work

For this project, I only compared a model from scratch with a single pre-trained model, the VG166 (characterized by small 3x3 convolutional filters with depth of 16 layers). However, Keras comes with other ConvNet models built-in and thus could have also been tested. Future models to be experimented with include Xception, VGG19, ResNet50, InceptionV3, InceptionResNetV2, MobileNet, DenseNet, NASNet (full list replicated here: https://keras.io/applications/).

Secondly, gathering and training on a larger image data set is another opportunity for further work. Instagram and Facebook could be potential sources for a trove of labeled data for this application.

8. Conclusions and Recommendations

8a. The Transfer Learning Advantage

Building a CNN from scratch illustrated how complex and computationally demanding computer vision tasks can be. With my best efforts, my CNN without pre-trained weights was able to achieve an accuracy of 39.6%. The VGG16 model with pre-trained weights was able to achieve a substantial improvement but only with the aid of ImageNet weights which had been trained for weeks on multiple GPUs. The accompanying research with the VGG16 article also cites how susceptible to stalling training a model can be random weight initializations. Through this process, I have a deeper appreciation for the advantage (not to mention time savings) of utilizing a pre-trained model over building a model without pre-trained weights.

8b. State of Computer Vision

Visually identifying dog breeds can be a difficult task for humans and still poses a challenge for computer vision. However, it is remarkable how far computer vision has come and what is capable of. Even with smaller datasets and a base understanding of neural networks, it is possible to build a decent model for an array of applications due mostly to the ability of Transfer Learning and advancement of user-friendliness of open source libraries.

8c. Deployment into Production

Though the model performed impressively at a human-scale, being accurate 58.9% of the time on the test images is not a level of performance to be deployed for systematically auto-labeling images (the initial goal of applying CNNs). Even so, the model can still be leveraged in its current form. For example, the model can attempt to auto-label for the purpose of higher level ad targeting and suggestions (for example, labeling pictures in a category of breeds). Another application of the model could be a suggestion engine for hashtags of the top 3 predicted breeds prior to posting (in which case the posted is assumed to be correctly labeling on the model's behalf).

9. Lessons Learned from this project

When possible, its highly advised to plot out the entire structure of the project and the desired results (such as accuracy metrics) and outcomes (including plots, confusion matrices and other visualizations). With this habit in place, it becomes easier to break down the larger problem and cement understanding of how to execute each of these sub-tasks. In my case, I found out too late that the loss/accuracy of a model isn't explicitly saved. Instead it needs to be assigned to a variable which then contains the "history" callback. As a result, I needed to re-run and re-train most of my models just to be able to extract a plot. Had I investigated this at the start, I could have saved myself hours of time.

More data is almost always better. Instead of spending a week trying to optimize a model and figure out different methods, I could have used that time to gather and label training data for myself. This is a common sentiment within the computer vision community that I learned first-hand.

For better accuracy and fine-turning, more computational power than what I possess is needed. Compared to a tutorial I followed along with, I was always about 20% short with respect to training accuracy. For instance, I could only have two fully-connected layers with 512 and 256 nodes, respectively whereas the tutorial had two layers of 2048 nodes. These fully-connected layers at the output of the pre-trained model helped the model better learn high-level, complex features (face, nose, colors, etc.) compared to my architecture.

10. Appendix

1. Image Data Two Ways

Pickling and converting in a 4-d numpy array

I initially utilized the Keras .flow_from_directory() method to pass in image data into the model for training. This method is in lieu of loading in 9600 images as a single tensor. Having a large data object in memory, compounded with the task of back propagation over a bevy trainable parameters, would not have been feasible on my computer. However, during the initial phases of building a Convolutional Neural Network from scratch, I was not able to attain over 1% accuracy on either the train or validation images using this method.

To get around this, I re-appropriated code and process to convert and crop the image files into a pickle file. The pickled files would then be converted into Numpy arrays, split into training, validation and test sets and then stored as a 4-d array (number of images, image width, image height, and the color channels).

For model training, these large arrays (~5GB for training and test sets) would be loaded into memory and passed into a .flow() method which passed in batches of the training data into the model.

Flow_from_directory method

The flow_from_directory method in Keras is a more intuitive method of training a model and cleverly infers class labels from the directories (eliminating the need to declare labels). This method is preferred for my application as I am dealing with a large image data set where each class contains relevant samples. This method also allows for an ImageDataGenerator processing pipeline which then can processed in batches. In other words, the pipeline iterates over a batch of images, processes the batch over respective parameters and feeds into the model (can be done for train, validate and test sets).

At the outset of project where I built Conv nets from scratch, I could not surpass an accuracy of 1-2% on both training and validation sets using the flow_from_directory method. However, this method only became effective when I implemented transfer learning and using pre-trained networks.

With respect to either method, the images were resized to 224 x 224 pixels while maintaining a RGB color channels. Additionally, as a normalization technique to offset overfitting and to streamline the training process, images were rescaled from 1-255 (color value of each channel) to 0-1.

The below table is a summary comparing the two methods:

	Pickling Data into Array Object	Keras Flow_from_Directory
Memory Intensive	Yes (>5GB)	No
Pass into .flow() method	Yes	Yes
Can be trained in batches	Yes	Yes
Can be fed into ImageDataGenerator?	Yes	Yes
Training Time	Longer	Shorter
Train/Validation Split	User-Defined	Can be user-defined or as a method parameter

2. Environment Set Up

For this project, I used my personal laptop which is a Lenovo Y50-70 with an Intel i7 @2.5Ghz, upgraded to 16 GB of RAM, and a NVIDIA GeForce GTX 860M GPU on Windows 10. Combatting thermal throttling was an issue when training models using GPU acceleration. If simulating this or a similar project, please be mindful as continued heat under constant load can be detrimental to the longevity of your system components. In my case, I used a laptop cooling pad in combination with a desk fan. Without these physical augmentations, I noticed substantially longer training and processing times.

I also installed Tensorflow and configured Keras to use the Tensorflow backend. On top of this, I configured Tensorflow to utilize my onboard GPU. This required the installation and configuring of the deep learning library from NVIDIA, cuDNN v7.0, and the CUDA Toolkit v9.0. It is also important to note that laptop GPUs have limited memory (2GB in my case) relative to their desktop counter parts. It was essential to set parameters to GPU usage as to not run into Exhaust Errors. I suspect this is a non-issue on newer cards on desktop.

3. Sources

1. Pet Industry Market Size & Ownership Statistics. http://www.americanpetproducts.org/press_industrytrends.asp

2. “The highly profitable, deeply adorable, and emotionally fraught world of Instagram’s famous animals”, Quartz. February 16, 2016. https://qz.com/612796/the-highly-profitable-deeply-adorable-and-emotionally-fraught-world-of-instagrams-famous-animals/

3. “20 Pets That Make Millions for Their Owners”, Huffington Post. July 8, 2015. https://www.huffingtonpost.com/gobankingrates/20-pets-that-make-million_b_7755792.html

4. “Inside the glorious and lucrative world of Instagram's famous pups”, Mashable. August 26. 2017. https://mashable.com/2017/08/26/instagram-famous-pets-money/#mv86uReL4mqw

5. “New BarkBox Study: Dog People Post About Their Dog on Social Media Six Times Per Week”, Bark.co. January 11, 2017. http://bark.co/barkgoodpartners/new-barkbox-study-dog-people-post-about-their-dog-on-social-media-six-times-per-week/