# **Machine Learning Engineer Nanodegree**

## **Capstone Report**

Shuyu Jia  
April 25, 2019

## **I. Define the Problem**

### **Domain Background**

In 2013, Kaggle hosted one of its favorite for-fun competitions: Dogs vs. Cats. [1] Everything has changed drastically in the field of machine learning, especially in deep learning and image classifications. Tracing back to 1998, when LeNet-5, a pioneering 7-level convolutional network was introduced, it was used for recognizing hand-written numbers digitized in 32 by 32 pixel greyscale input images. [2] In 2012, AlexNet significantly outperformed all the prior CNNs, winning the competition by reducing the top-5 error from 26% to 15.3%. Then all of a sudden, thousands of people devoted to building deeper and deeper CNNs, with higher and higher accuracy of course. Inception, VGG, ResNet are some famous CNN frameworks with very high accuracy. However, what remains unchanged is that CNN is still one of the most powerful tool in image classifications. In other words, convolutional layers are still quite efficient to extract spatial features on the pictures.

### **Problem Statement**

This capstone project is a binary classification problem. In this project, I will train a convolutional neural network (CNN) to differentiate pictures of dogs and cats, using the training dataset provided by Kaggle. Then for each image in the test set, I will provide a probability that image is a dog. The final result can be measured by the log loss. A smaller log loss is better.

### **Datasets and Inputs**

As I mentioned above, both the training set and the testing set are provided by Kaggle. The train folder contains 12500 dog images (labeled from dog.0 to dog.12499) and 12500 cat images (labeled from cat.0 to cat.12499). Each image in the folder has the label as part of the filename. Without loss of generality, I put the last 2500 dog images and last 2500 cat images into the validation folder, and the training folder remains 10000 dog images and 10000 cat images, so that it maintains a 4:1 training versus validation ratio, which is pretty reasonable.

The test folder contains 12500 images (labeled from 1 to 12500), named according to a numeric ID. For each image in the test set, I will predict a probability that the image is a dog. Specifically, 1 represent a dog, and 0 represent a cat.

### **Solution Statement**

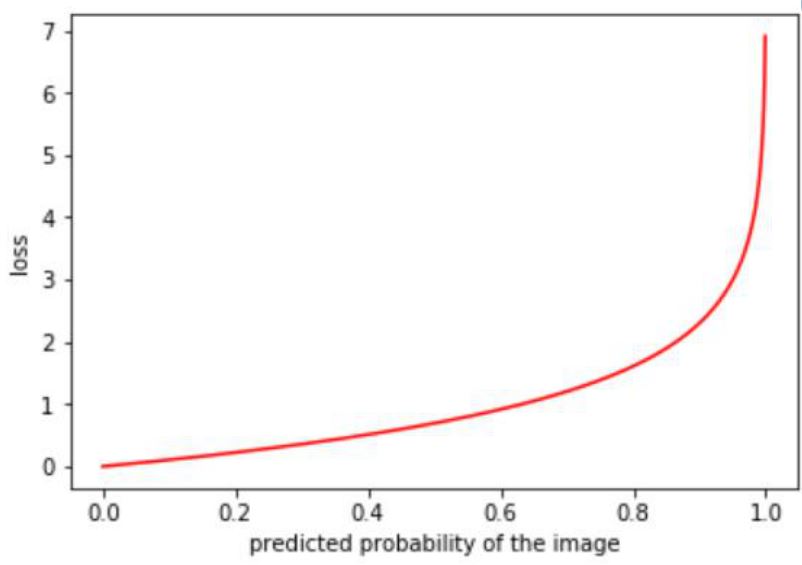
I will use transfer learning technique with PyTorch to solve this problem. The base pre-trained model I will use is ResNet-152. Firstly, I will replace the out feature in the last fully connected layer from 1000 neurons to 2 neurons, and freeze all parameters except the classifier. Then I will train the parameters of the classifier.

### **Evaluation Matric**

I will use the same evaluation matrix as in the Kaggle Dogs vs. Cats Competition. Our result is going to be scored on the log loss derived from maximum likelihood estimation (MLE). The formula is as follows:



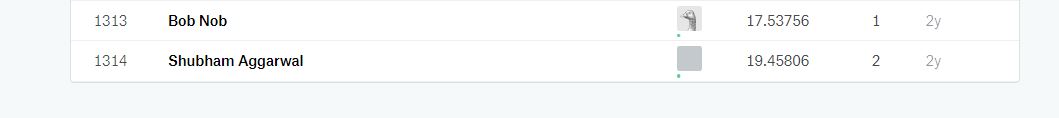
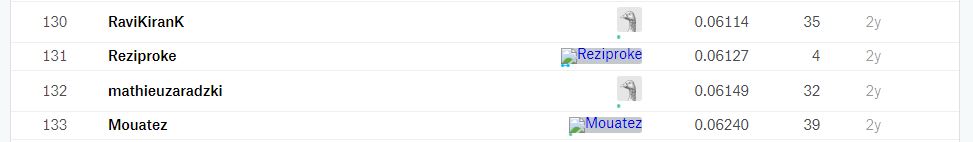
Here n is the number of images in the test set, y\_i hat is the predicted probability of the image being a dog. Note that y\_i is 1 if the image is a dog, and y\_i is 0 if the image is a cat. We also use natural logarithm based e. [1] The following graph shows how the log loss changes for every predicted probability of an image labeled 0.



From the graph we can see that log loss is quite good at measuring the performance of our model. Specifically, if the predicted probability is very close to the label, the loss is very small. On the other hand, if the predicted probability is very close to the opposite label, then the loss becomes huge.

### **Benchmark Model**

There is no benchmark model in this capstone project. However, our performance can be measured by comparing our log loss to the public leaderboard. The public leaderboard ranks every team by the log loss, so that we can compare our final log loss to the leaderboard and see how well we are doing. A smaller log loss is better. There are 1314 team results on the public leaderboard, and my goal is to reach the top 10% of the public leaderboard (top 131). The log loss is approximately 0.06127 within the top 10% performance. In other words, our goal is trying to make our final log loss less than 0.06127.

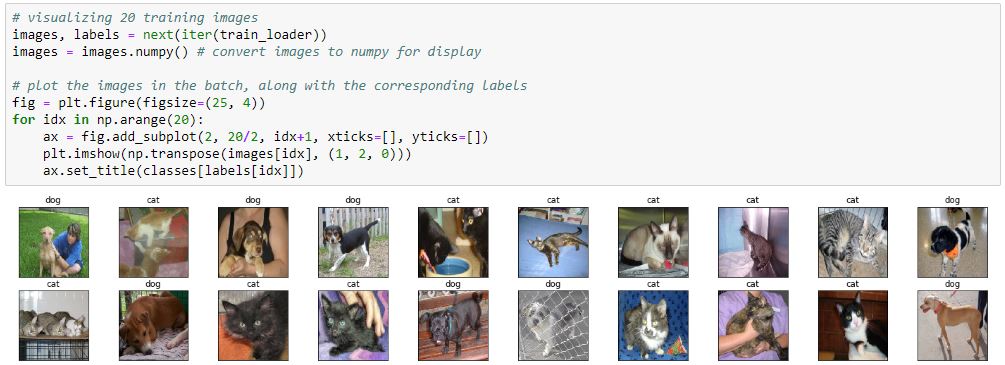


## **II. Analysis**

### **Data Exploration**

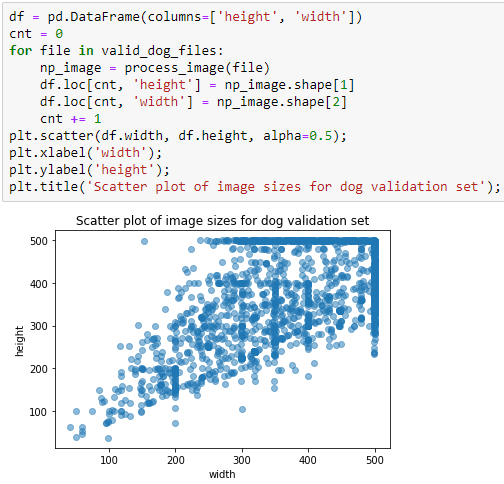


As we discussed in the last section, we have a total of 20000 training images, with 10000 each for cat and dog, and a total of 5000 validation images, with 2500 each for cat and dog, and a total of 12500 test images needed to be predicted.



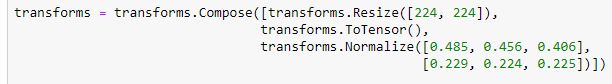
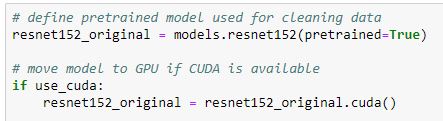
As we visualize some of the images in the training set, we found out that our dataset is not very clean. For most of the time, our dogs and cats occupy the whole spatial image, while some of them are pretty small in the images. There are different kinds of backgrounds behind the dogs and cats, such as lawns, nets, blankets, or even humans; some dog and cat images are partially covered by blankets or something else, others are taken in front of a net; they are all noise in this classification problem. The position of the cats and dogs in the images is not fixed as well, but most of them are located in the center of the image.

Moreover, I also created a scatter plot of image sizes for dog validation set, so that we can see how image sizes are distributed. From the scatter plot we can see that all images have a size less than or equal to 500 pixels by 500 pixels. A few images have a size less than 100 pixels by 100 pixels, but there are not a lot of them. Thus, we can say that most of the images are decently clear so that we do not have to perform image sharpening.

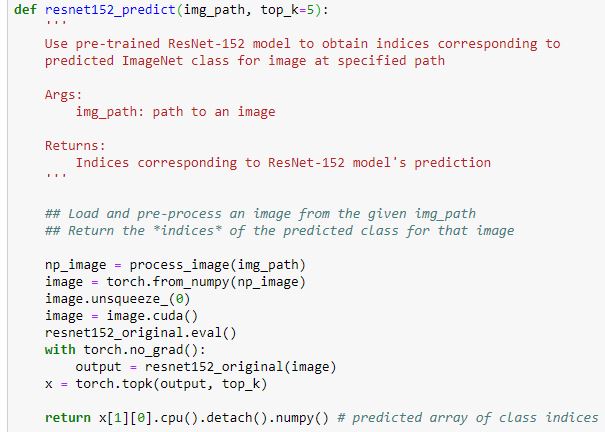


### **Data Cleaning**

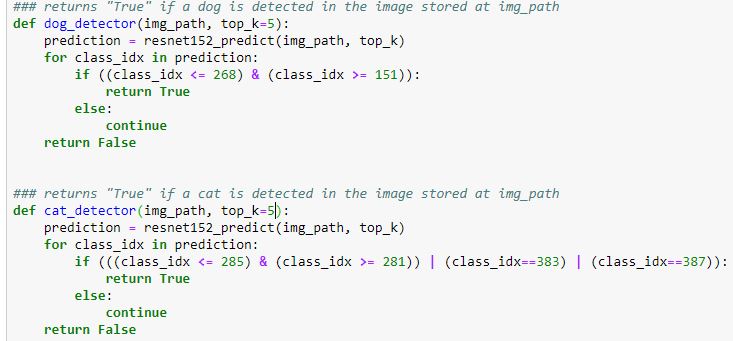
Not all images in our original dataset are appropriate to use to train our model, and we need to remove those images that are not qualified. To do this job systematically and to make sure we do not miss any, we can use one of the pre-trained model for the ImageNet classification problem inside the “torchvision” package. Among all the listed models, ResNet-152 has the lowest top-1 error (21.60) and the lowest top-5 error (5.94) [4], so I decided to use it for cleaning our data.



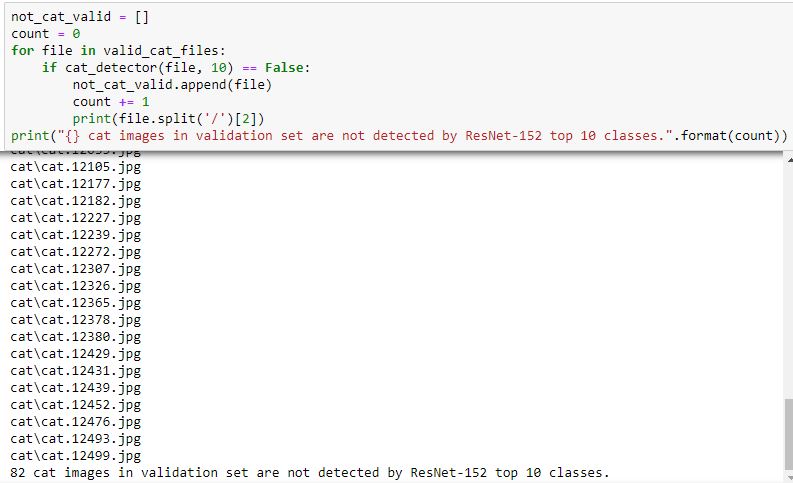
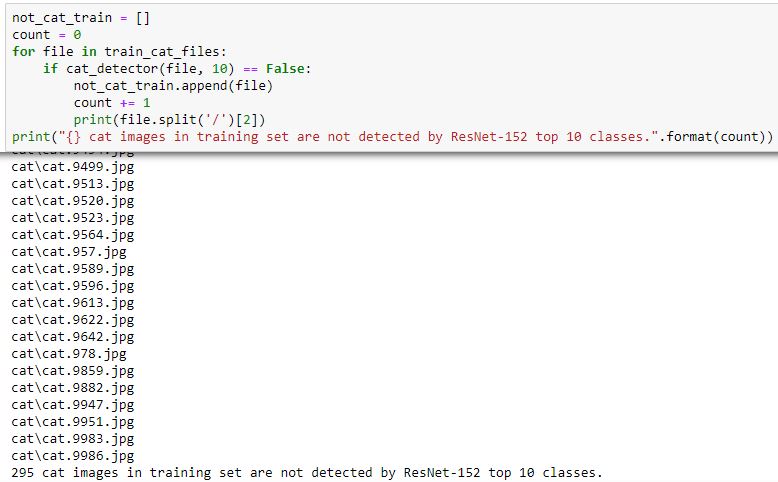
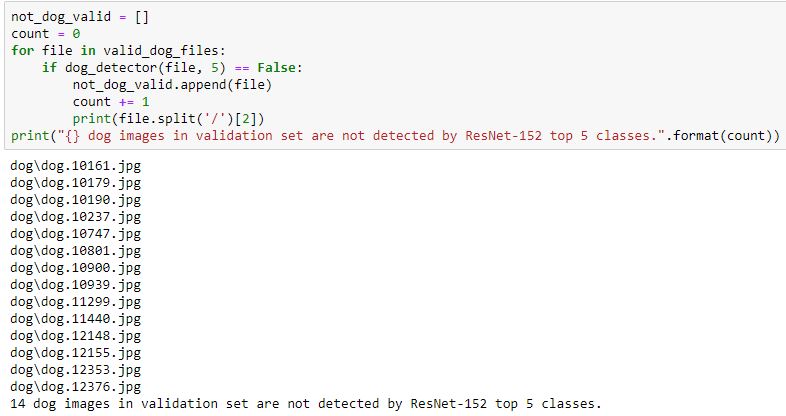
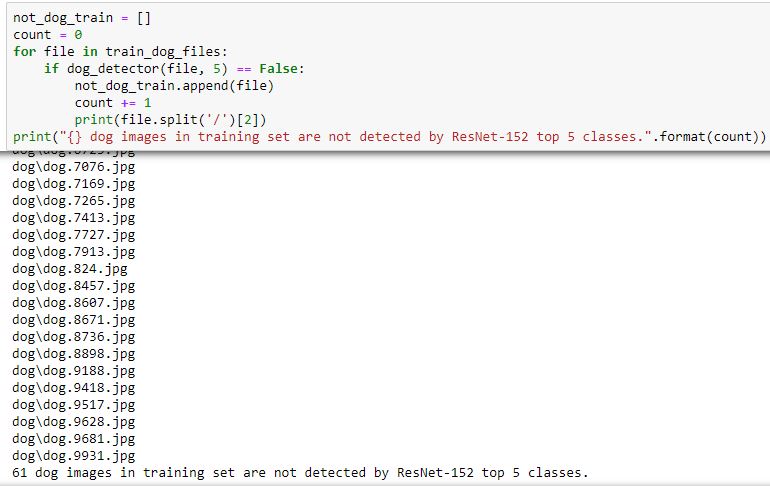
I loaded the ResNet-152 pre-trained model and moved it to GPU if available. In PyTorch, all pre-trained models expect input 3-channel RGB images normalized in the same way, and both the width and the height of the images are expected to be at least 224. The images have to be loaded into a range of [0, 1] and then normalized using mean equals [0.485, 0.456, 0.406] and standard deviation equals [0.229, 0.224, 0.225]. [4] Thus, we used a series of transforms to resize the images into 224 by 224, to transfer the images into tensors, and to normalized them.



The “resnet152\_predict” function takes an image path and returns an array of top k predicted class indices (default is top 5).



The pre-trained ResNet-152 model is originally used to classify 1000 image categories on ImageNet. Among the 1000 categories, 118 of them (index 151 to 268) are dogs and 7 of them (index 281 to 285, 383, 387) are cats [3]. The “dog\_detector” function takes a image path and returns “True” if a dog is detected in the image by the top k classes of ResNet-152 pre-trained model, so does the “cat\_detector” function.



As a result, there are 61 dog images in training set and 14 dog images in validation set are not detected by ResNet-152 top 5 classes. On the other hand, 295 cat images in training set and 82 cat images in validation set are not detected by ResNet-152 top 10 classes. I checked all of the images manually and found out 79 images that are not reasonable to use as training or validation images. I classified all unqualified images into 7 categories and let me present some of them below:

1. Two images out of 25000 are falsely labeled. “cat.4085” and “cat.12499” are both dog images but they are labeled as cat. This ratio is actually acceptable, and I can just delete them from the dataset.
2. Some images contain both dogs and cats, which is definitely not appropriate to use in this binary classification problem.



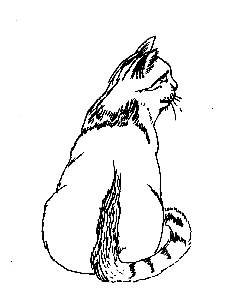
1. Some images are not available anymore on the internet, and they are replaced by a placeholder.



1. Some company logos are relevant to cat and dog but they are not even close to dog and cat images.



### Some hand drawing pictures or cartoons that are not real images:

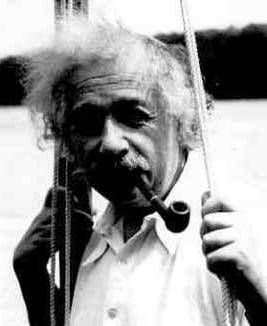
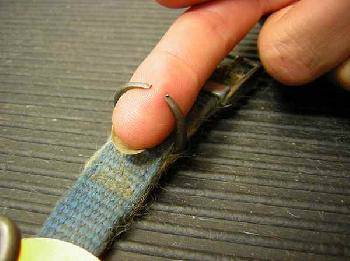
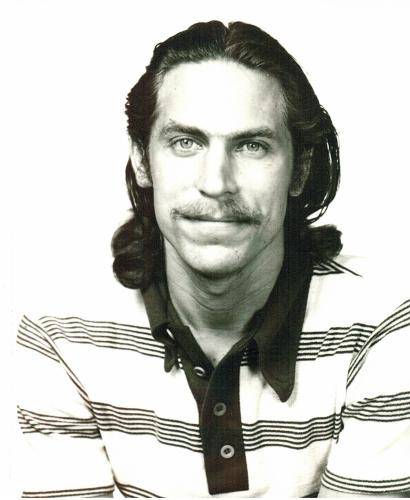


### Some part of animals like claws, legs or even belly, which is not representative of a cat or dog image. Therefore we should not use them.

### 



### Other pictures that are not appropriate in this problem.



### **Algorithms and Methods**

To solve this problem I am going to use transfer learning. Transfer learning is a machine learning technique where a model trained on one task is re-purposed on a second related task. The ResNet-152 model was trained to categorized images on ImageNet. And since dogs and cats are within the 1000 categories on ImageNet, the problem we are trying to solve is actually related to what ResNet-152 was trained for. On one hand, the ResNet-152 architecture is proven to work really well on image classification tasks. On the other hand, because of the relatedness, the pre-trained parameters are going to be so much better than some random initialized weights. However the problem we are solving is a binary classification problem, so that we need to replace the fully-connected layer with 2 neurons for the out feature. Then we only need to train the fully-connected layer to get a very decent result.

ResNet-152 contains 151 convolutional layers and 1 fully-connected layer, and each convolutional layer is followed by a batch normalization layer. Those 151 convolutional layers are grouped by 3, so that we have 50 groups of them with one single convolutional layer at the very beginning. There is a ReLU (Rectified Linear Unit) activation layer after each group (a batch of 3 convolutional layers). At the very beginning the convolutional layer is followed by a batch normalization layer, a ReLU layer, and a max pooling layer. The first 3 groups are named as “layer1”; the next 8 groups are called “layer2”; the next 36 groups are noted as “layer3”; and the last 3 groups are known as “layer4”. In the very end followed by an average pooling layer and a fully-connected layer.

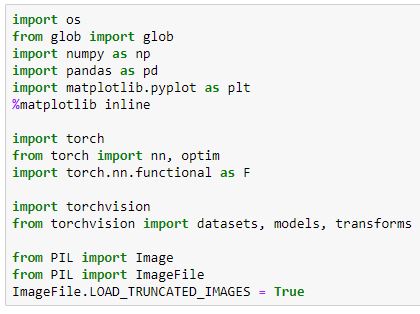
There are 2 advantages by using a series of convolutional layers. Firstly, convolutional layers use less parameters than fully connected layers (or dense layers), so it consumes less computing power. Secondly, convolutional layers accept matrices as inputs, so we do not throw away all the spatial information like dense layers do. Batch normalization layers improve the speed, performance and stability of the neural networks, since it normalize the hidden layers by adjusting and scaling the activation layers, just like we did to the input layers. Max pooling layers effectively reduce the dimentionality in each step, and at the same time we can keep all the useful information. By adding a global average pooling layer in the end, we further reduce the number of parameters used in the last fully connected layer.

Having such a deep neural network, ResNet architecture effectively solve the vanishing gradient problem by calculating gradient across multiple convolutional layers, instead of every convolutional layer as in the traditional architectures. Using ReLU activation layers also prevents the gradients from being vanished, because it has a derivative of 1 for the positive gradients.

In this problem I used categorical cross-entropy as the criterion to calculate loss. In PyTorch the categorical cross-entropy loss contains both negative log likelihood loss and the log softmax function, which matches the evaluation matric used by Kaggle. We also used stochastic gradient descent (a.k.a. SGD) as the optimizer. Instead of using the whole training data to update the weights once, SGD only takes a batch of images. This will efficiently use the computing power of our GPU without sacrifice much accuracy to the result. The specific number of one such batch is called the batch size, and I set it as 256 in this problem.

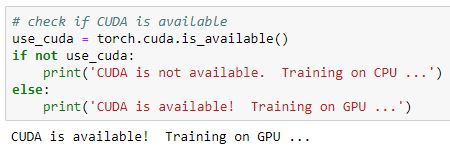
## **III. Solving the Problem**

### **Load Packages**



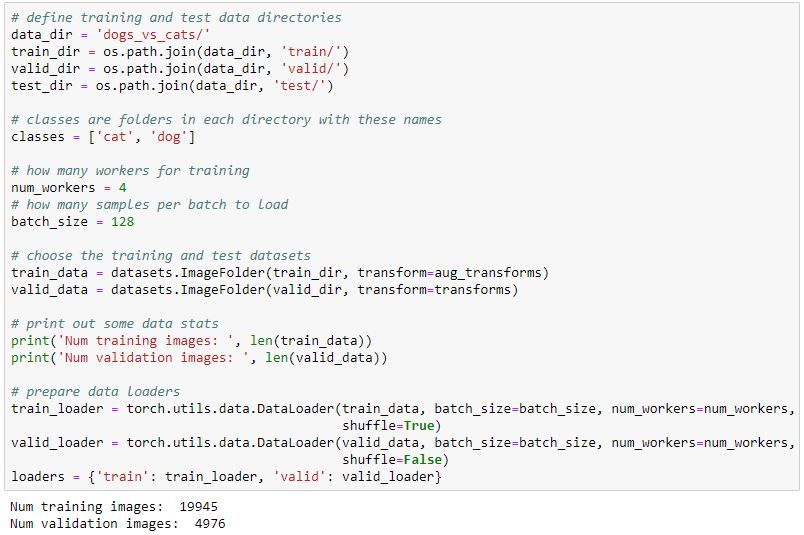
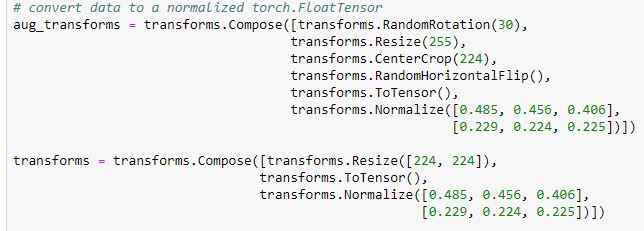
In this problem I utilized 8 third-party packages. Specifically, “os” is used to create path of datasets; “glob” is used to create files in data cleaning; “numpy” is used to deal with arrays; “pandas” is used to read in and manage submission files; “matplotlib” is used to create visualizations; “torch” is used to train our model; “torchvision” is used to preprocess our data; and “PIL” is used to process and manage raw images.

### **Check the Availability of GPU**

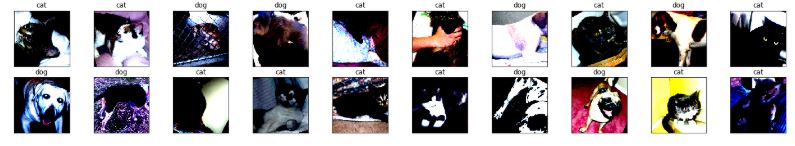
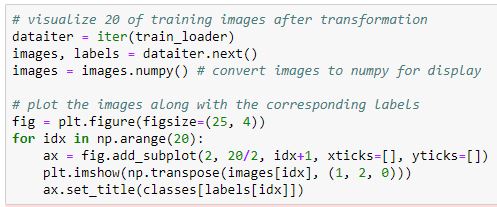


Since we are going to use transfer learning technique to solve this problem, we need to use GPU to greatly reduce our training time. The above cell checks the availability of GPU and prints out the result.

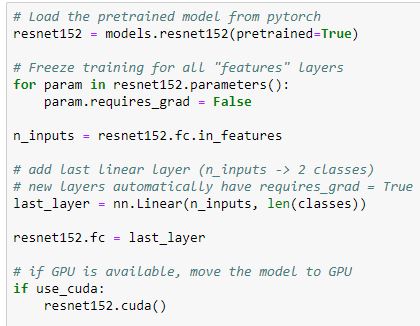
### **Load Data**



After removing all outliers in our dataset, there are 19945 training images and 4976 validation images left. Then I created both train loader and validation loader. In the mean time, I augmented the training data by random rotation and random horizontal flip. I also visualized some of the training images as follows after data augmentation and data normalization:



### **Define the model**



Firstly, I loaded the pre-trained ResNet-152 model from PyTorch and froze all parameters, so that we can only train the parameters we want to train. This time we are going to use it to classify dogs and cats, so there should only be 2 outputs in the last layer. Thus I changed the last layer with 2 outputs and moved it to GPU if available.

### **Train the model**

I wrote a function called “train” for our convenience later. This function takes in the number of epochs we would like to train, data loaders, model to train, optimizer to update weights, criterion to calculate loss, GPU’s availability, saved path of best parameters, as well as the current validation loss. The training part of the function runs forward pass and back-propagation to our training set, then updates the weights. The validation part of the function runs only the forward pass to our validation set, and calculates the validation loss. If the new validation loss is less then the previous minimum, then we set the new validation loss as the minimum loss and save the model. For every epoch there will also be printouts recording the current training loss, validation loss, if the validation loss has decreased, and if the new parameters are saved. The “train” function is showed as follows with detailed comments.



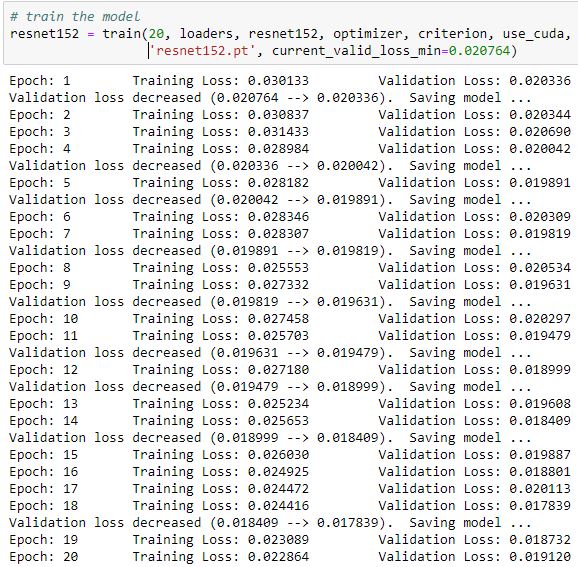
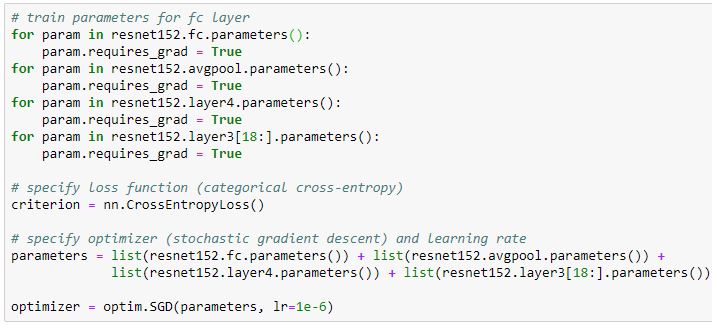
I opened the fully-connected layer only and trained 20 epochs, getting a validation loss of 0.020764, which is decently low but we need further improving our model.

### **Improvements (Fine-Tune)**

I did the following trials to further improve our model:

1. I opened the average pooling layer and the “layer4”, and trained 20 epochs with a learning rate of 1e-3.
2. When the validation loss is not decreasing anymore, I opened the last 6 groups in the “layer3” (noted as layer3[30:35]), and trained 20 epochs with a learning rate of 1e-4.
3. When the validation loss is not decreasing anymore, I opened another 6 groups in the “layer3” (noted as layer3[24:35]), and trained 20 epochs with a learning rate of 1e-5. Since my GPU only has a memory of 11 GB, I changed my batch size from 256 to 128. Otherwise there would be an error message complaining the CUDA is out of memory.
4. When the validation loss is not decreasing anymore, I opened another 6 groups in the “layer3” (noted as layer3[18:35]), and trained 20 epochs with a learning rate of 1e-6. Batch size remains 128 here.

When we goes deeper and deeper in the model, the filtered features are becoming simple and simple so that there is no point to go deeper anymore. Those deeper features can be used in any images. Moreover, any layer beyond 18 groups of “layer3” is going to run out of memory in my GPU again. If we go deeper, I have to change my batch size to 64, which is not decent any more. The weight updates are getting worse and worse with smaller batch size simply because of the randomness. Therefore, I stopped fine-tuning until the validation loss is not decreasing anymore in step 4. I listed an example of my fine-tuning records as follows:



My final validation loss reached a minimum of 0.0141.

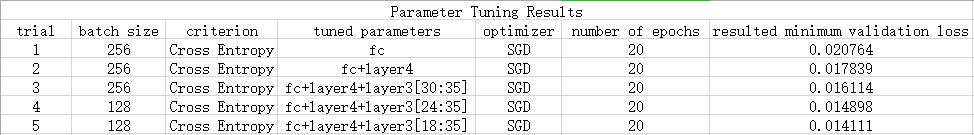


Before we tested our model, we need to load the model with the lowest validation loss.



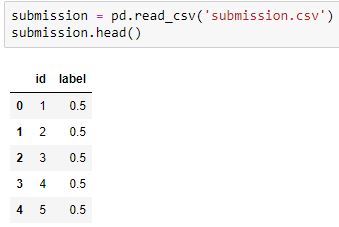
## **IV. Results**

### **Parameter Tuning Results**

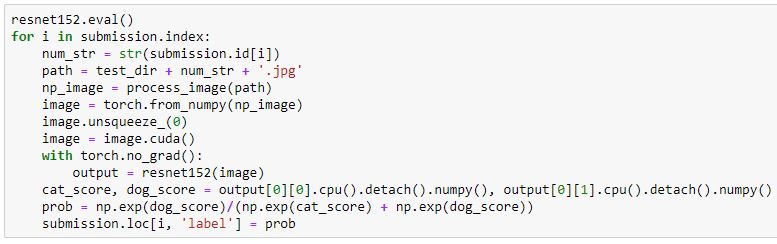


The above table recorded the parameters and the corresponding resulted minimum validation loss for each training trial. We can see that the validation loss is getting smaller and smaller after each training trial, indicating that our model parameters are gradually optimized along the way.

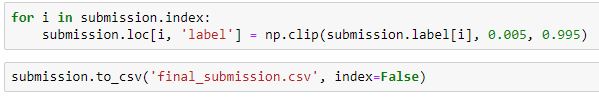
Kaggle provides us a sample submission. Let us firstly load the file using “pandas” library.



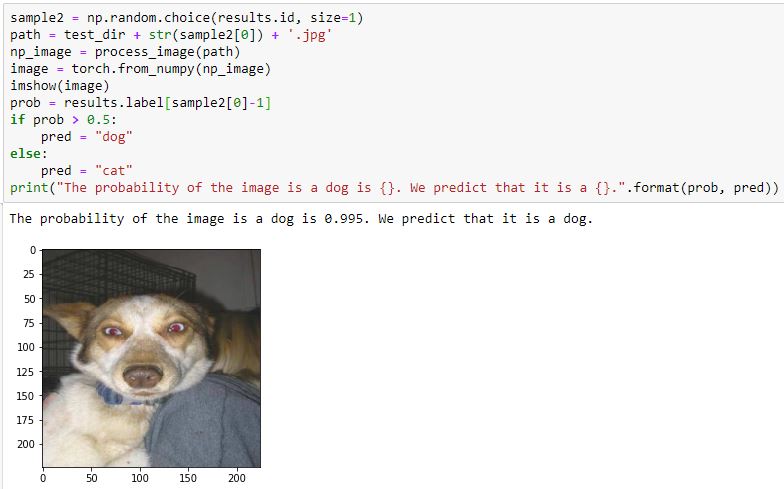
I turned the model into evaluation mode and run forward pass to every test image. For the two scores in the output, I used the softmax formula to get the probability of dog, and used it to replace the original labels.



Next, I clipped every probability into a range of [0.005, 0.995]. This is because we can never be too certain of something. Predicting some image has a dog probability of 1 is not appropriate. Lastly, I saved the dataframe as my final submission.



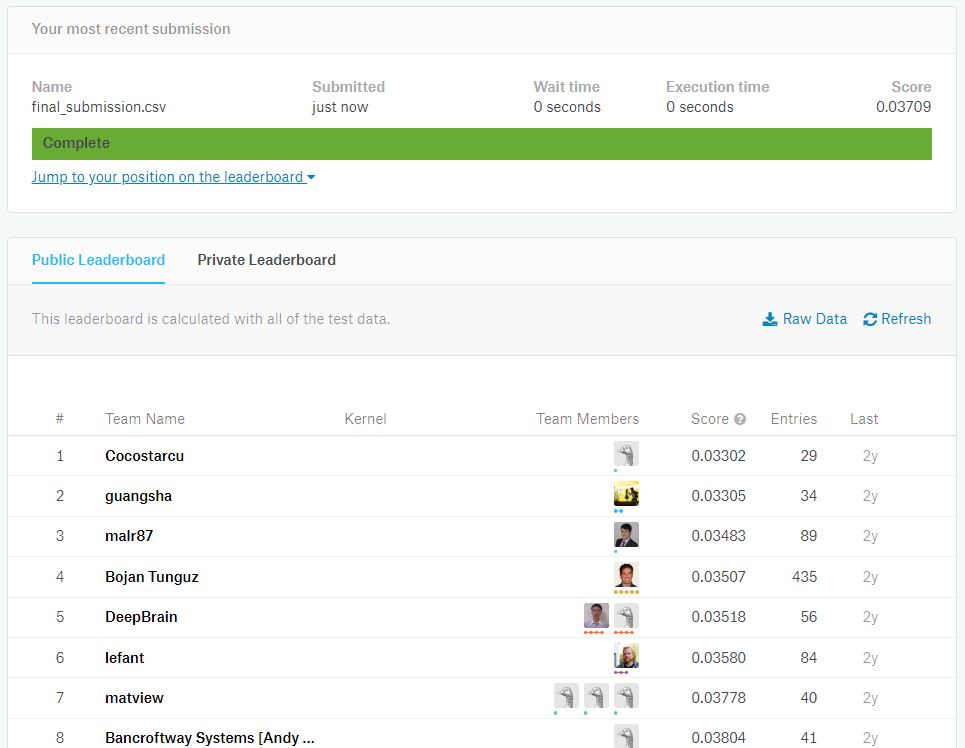
Before we upload the results to Kaggle, I created a few visualizations in order to see if our model behaves properly.



We can also see that among all the probabilities our model predicted, only 91 out of 12500 are between 0.1 and 0.9, which indicates that our model gives very strong predictions to most of the images. It seems that out model is pretty decent and robust.



After we uploaded it to Kaggle and compared to the public leaderboard, we can see that our log loss is 0.03709, which ranked number 7 on the public leaderboard. This reaches our goal (top 10% in the public leaderboard).



## **V. Reflection & Improvement**

### In this capstone project I successfully using transfer learning technique to train a model that decently classifies cat and dog images. The final score is ranked number 7 on the public leaderboard. I detailed recorded all the training process, and commented all of my code carefully. I like to fine-tune the model, adjusting the learning rate and batch size, and controlling the depth in the training process at the same time. When I saw the validation loss is getting smaller and smaller, I feel quite exciting. However it is quite difficult sometimes. It is going to take forever to train if the learning rate is too small, while the validation loss will not converge if the learning rate is too large. Although larger batch size tends to give us better weight updates, we should be careful since there is a limit for the GPU’s memory. So there is really more of an art than a science here. Our final model tends to give strong predictions to a certain image, which shows robustness.

There are certainly some improvements with which I can make the model even better. Data-wise, there are certainly more ways in PyTorch library to perform data augmentation other than random rotation and random horizontal flip. We can even write our own function to augment the data. Moreover, I can gather more dog and cat images online to build a even larger training set and validation set. Method-wise, I am only using a single model prediction. I can also train multiple robust models based on other well-performed models on ImageNet classifications such as Inception V3 and DenseNet-161. [4] Merging all 3 models and takes the average or simply just let them vote is going to give us even higher accuracy and even smaller loss. Furthermore, we can also adjust the dropout rate in order to prevent over-fitting. Tech-wise, a GPU with higher memory capacity is going to allow me to train even deeper parameters or to train with a higher batch size. All of these potential improvements are going to allow me go further in the future.

****References****

[1] “ Dogs vs. Cats Redux: Kernels Edition.” *RSNA Pneumonia Detection Challenge | Kaggle*, 2017, [www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/data.](http://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/data.)

[2] Das, Siddharth. “CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and More ....” *Medium.com*, Medium, 16 Nov. 2017, medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5.

[3] Yagnesh. “Text: Imagenet 1000 Class Idx to Human Readable Labels (Fox, E., & Guestrin, C. (N.d.). Coursera Machine Learning Specialization.).” *GitHub*, Yagnesh, gist.github.com/yrevar/942d3a0ac09ec9e5eb3a.

[4] “Torchvision.models” *Torchvision.models - PyTorch Master Documentation,* Torch Contributors, 2018, pytorch.org/docs/stable/torchvision/models.html