# DOG BREEDS CLASSIFICATION USING NEURAL NETWORK

A PREPRINT

**Yijun Mao**
yijun.mao@duke.edu
**Yiteng Lu**
yiteng.lu@duke.edu
**Yihao Hu**
yihao.hu@duke.edu

**Wenge Xie**
wenge.xie@duke.edu
**Shixing Cao**
shixing.cao@duke.edu

August 4, 2019

**ABSTRACT**

Dogs come in various shapes, sizes and colors. However, it is difficult for people to recognize the breed of an unfamiliar dog. In this article, we used deep learning approach to train a dog-classification model. Given an image of a dog, the model will predict the breed of the dog. If an image of a human is supplied, the model will identify the most resembling dog breed of that person. The trained model can be used by a web or mobile application to process real-world, user-supplied images. We compared the result of our model with off-the-shelf ResNet50. The evaluation results turned out that our model achieve relatively good accuracy, which is around 89 %.

***K*eywords** Dog Breeds · Classification · ResNet 50

## 1 Introduction

In this project, we built and trained a convolutional neural network model, specifically ResNet50 from scratch, and compared its performance with a pre-trained ResNet50 model in detecting dogs in images.

### 1.1 Domestication History

There is a great deal of debate on research into the events surrounding wolves' first domestication into dogs by humans during prehistory. Via genetic analysis of extant populations of both groups, evolutionary biologists hold the view that the event occurred somewhere in Eastern Europe or Asia between eleven and sixteen thousand years ago [1]. The hypothesis indicates a date earlier than both the invention of agriculture and the domestication of any other wild species of animal. Hence, our special relationship with *canis lupus familiari* extends back farther in time than almost any other as well as our organized societies and technologies. The astounding diversity of domestic dogs alive today reflects the chronological depth of this relationship. Furthermore, the household canine is the most morphologically and genetically diverse single species of animal on Earth. There are substantial breeds of dogs which present hugely varying physical and psychological traits ranging through color, size, shape and behavior.

### 1.2 Breed Identification Problem

This great variety makes trouble for people who would like to know the breeds of a dog seen in daily life. Walking down the street or sitting in a coffee shop, one might see a friendly, lovely dog and wonder at its pedigree. In many cases, it may not be possible to ask an owner about the breed, and chances are that the owner themselves will be either unsure about giving an accurate answer. Unless the dog falls into one of a few very widely known and distinctive breeds such as the golden retriever, Welsh Corgi and Siberian husky etc., it is difficult to tell the exact breed of a dog without systematical research and analysis.

### 1.3 Deep Learning in Image Classification

Deep learning is widely used for image recognition nowadays. The patterns in different images are complex, and the decision boundaries are always highly nonlinear for real-world images. Deep learning methods are excellent fit for these requirements. Convolutional Neural Network (CNN) works as a circuit which is made by neurons. A neural network is a cascade of layers. CNN is commonly used for image recognition because it automatically extracts lower level features from the origin image and build higher level ones in hierarchies. Transfer learning is also used in training, which saves a lot of time for using the pre-trained weights. For dog breed classification, we use Residual Network (ResNet) which is a specific type of CNN.

## 2 Dataset

The full dataset used by this project contains 8361 images with 133 categories of dogs. The data is separated into three folders for training, validation, and testing, which respectively consists of 6680, 845 and 836 image samples. Each image in the dataset is annotated with the breed of that dog.

### 2.1 Preprocessing

In fact, having only 6680 images for training is not enough to learn an elegant deep neural network. It would not be able to learn generic enough patterns off this dataset to classify different dog breeds. Chances are that it will overfit to the small amount of training examples which results in low test accuracy. There are two possible approaches to mitigate the lack of training examples:

1. Merge dog dataset with another bigger dataset with images (i.e ImageNet).

2. Data Augmentation.

The first method is not practical because of extensive efforts spent on training the large dataset. The second is more viable, since training process can be executed on the original dataset by adding several transformation layers on neural network model. Data augmentation refers to randomly changing the images in ways that do not impact their interpretation such as flipping horizontally and vertically, zooming in and out, varying contrast and brightness, and many more.

### 2.2 Data Augmentation Method Applied in Our Datasets:

- Resize(256)
- CenterCrop(224)
- ToTensor()
- Normalize()

CNNs require input images to be converted into 4D tensors, so transformation is needed for the image data.

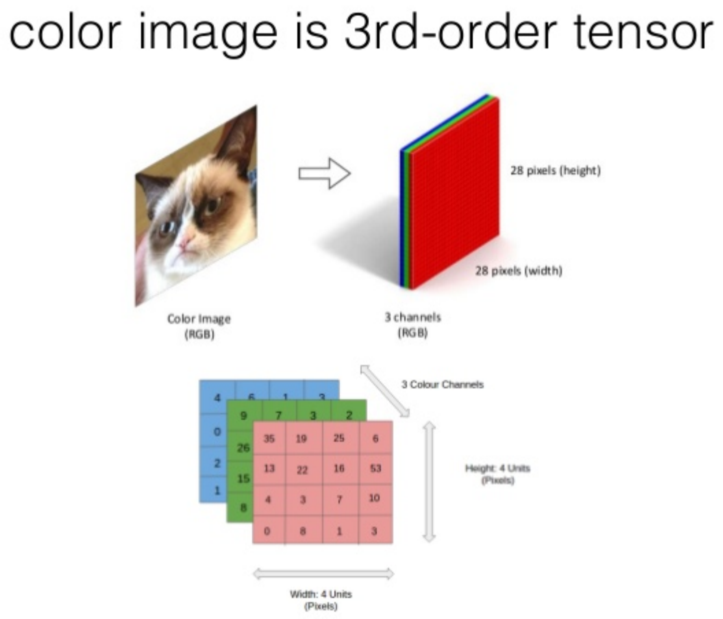


Figure 1: Tensor1

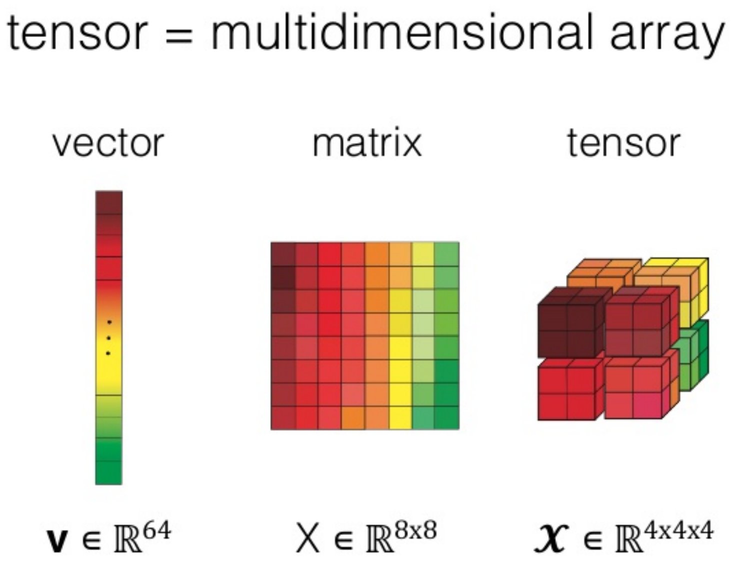

tensor = multidimensional array
vector
matrix
tensor
$\mathbf{v} \in \mathbb{R}^{64}$
$X \in \mathbb{R}^{8x8}$
$\mathcal{X} \in \mathbb{R}^{4x4x4}$

Figure 2: Tensor2

## 3 Neural Network Model

### 3.1 Working Principles of ResNet50

We decided to use Convolutional Neural Network for our classification task since it's the state-of-the-art technique in scope of image recognition and classification. Specifically we selected ResNet 50 since it's not too difficult to implement and it has proven to perform well in several image classification related scenarios.

ResNet was first developed and introduced into image recognition and classification contests by Kaiming He. The greatest advantage of ResNet is that it could utilize the depth of neural networks to enhance the performance significantly, which was impossible in previous networks. In fact, some of the ResNet structures can reach a depth of as deep as 1000 layeres. A question may be raised is why do we bother to develop a new network structure like ResNet, but not simply extend the layers of any existing neural networks to enhance the performance. This is due to a phenomenon called degradation of deep network, in which the performance of network shows no improvement and even degradation when the depth of the network increases. Kaiming has analyzed this phenomenon through a scenario as following. Assume there are two networks, the second one is deeper than the first one. We have known the appropriate weights for the first network through training. For the second network, if the beginning layers are exactly the same as the first one and all the following layers could learn weights equal to one, then the performance of the second network should not be worse than the first one. However, this is usually not the case and we find it's hard to make the deep layers learn weights equal to one. To solve this difficulty, Kaiming has proposed to forward the input of a block of layers to its output so that the layers are now expected to learn weights of zeros instead of ones. This is usually more approachable for networks. Figure 3 demonstrates the working principle mentioned above of a residual block of ResNet.

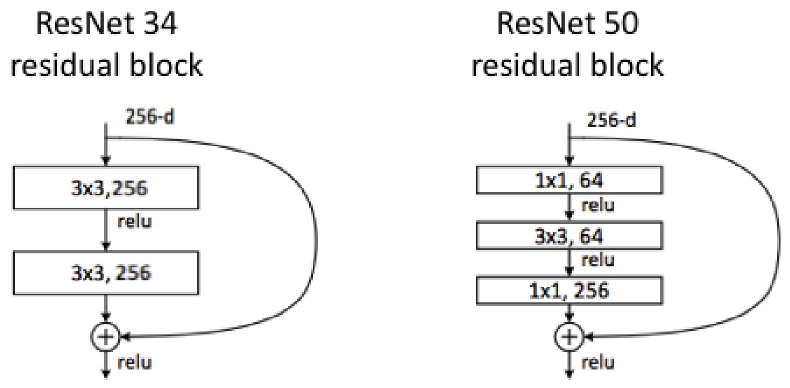
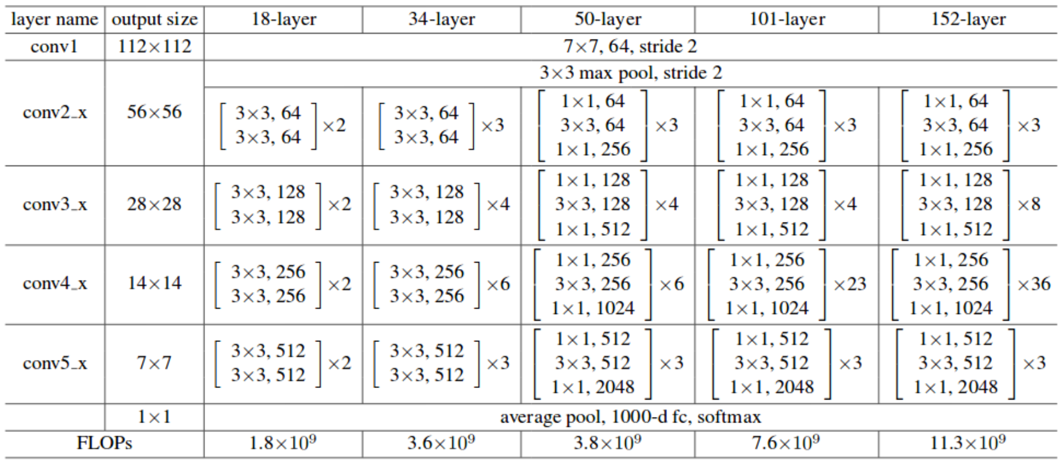


Figure 3: Working principle of residual block

## 3.2 ResNet50 Structure

There are several variants in ResNet family. From the shallowest ResNet18 to ResNet34, ResNet50, to deeper ones like ResNet101, and ResNet152. We mainly focus on ResNet50 in our project due to the reasons mentioned above.The basic building blocks of ResNet50 are a series of 3-layer blocks called bottleneck. This kind of structures can also be found in other ResNet models with more layers than ResNet50 such as ResNet101 and ResNet152. Actually the only difference among ResNet50, ResNet101, and ResNet152 are the different numbers included in various 3-layer building blocks. According to the original paper, there are 4 different types of bottleneck blocks, each of which consists of three different convolutional layers. After each of these convolutional layers, except the last one in this 3-layer block, there is one batch normalization layer followed by one ReLU layer. The situation for the last convolutional layer is a little bit different. After the last convolutional layer, first a batch normalization layer is inserted, then the input to the first layer of the three-layer block is added to the output of this batch normalization layer. Note that the dimension of input is different with that of the output of batch normalization for the second repetition of each type of three-layer block. A detailed structure scheme of ResNet 50 can be found in Figure 4.



| layer name | output size | 18-layer | 34-layer | 50-layer | 101-layer | 152-layer |
|---|---|---|---|---|---|---|
| conv1 | 112×112 | 7×7, 64, stride 2 | | | | |
| conv2_x | 56×56 | 3×3 max pool, stride 2 | | | | |
| | | $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$ |
| conv3_x | 28×28 | $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$ |
| conv4_x | 14×14 | $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$ |
| conv5_x | 7×7 | $\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$ |
| | 1×1 | average pool, 1000-d fc, softmax | | | | |
| FLOPs | | $1.8\times10^9$ | $3.6\times10^9$ | $3.8\times10^9$ | $7.6\times10^9$ | $11.3\times10^9$ |

Figure 4: Structures of different ResNet Models

In this project, we compared the classification results of applying a built-in ResNet50 model with those of a ResNet50 model build by ourselves using PyTorch. PyTorch is a powerful open-source Python package for machine learning. It was first developed by Facebook. We decided to use PyTorch since it's easier to get started than Tensorflow but offers all the functionalities Tensorflow could offer. There are mainly two reasons we built a ResNet50 model by ourselves apart from calling the existing one in TorchVision. The first is to learn how to implement a CNN model from scratch in PyTorch. It is good practice to cultivate the ability to implement a model from a research paper, especially the classic models like ResNet. The second reason is that because we build a ResNet50 model from scratch and we do not apply any transfer learning techniques on training it, we have to train the model from beginning. Thus, we can have a much better idea of how training works, how to initialize the weights and biases in the network, and how pre-training would affect the performance of the network. When building the ResNet50 model in PyTorch, we define two classes called "BottleNeck" and "ResNet" in which the first one defines the bottleneck-shaped building blocks while the second one builds up the whole model by calling the basic building blocks in the "BottleNeck" class combined with other layers such as max pooling, average pooling, and fully connected layer. We found dimension mismatch would cause huge pain in PyTorch, especially for a complex network such as ResNet. Also high attention should always be paid to match data types when doing operations like addition in PyTorch. Actually we've spent tons of time to debug before finally got the codes running due to unawareness of these two problems.

## 4 Training

### 4.1 Training Step

To train the ResNet50 model, we have some steps to follow.

First, we set up a termination threshold. We chose to consider the low variance of the validation accuracy as an indication of convergence. In practice, we set a threshold of 0.2 to be the standard deviation of every 5 epochs for the model we imported from the PyTorch library, and 0.004 for the model we built by ourselves. The reason we chose the threshold of our self-implemented model much smaller than the model from the library is that the model from the library has the built-in pre-trained weights, which means it can be trained from a better starting point than the model we built by ourselves. The pre-trained weights offered quicker convergence and larger increase of the validation accuracy among each epoch in the training process.

Second, we set up a second threshold which is the maximum number of epochs the training process could have. For the self-implemented model, we set the maximum epoch number to be 5000. For the model form the library, we set the maximum number of epochs to be 200. The reason for this setup is also because the model from the library converges much faster than the model we built by ourselves due to the fact that we did not use the pre-train parameters in the model we built by ourselves.

Third, We set up the loss function, optimizer, and scheduler we used in the training process. We used cross-entropy loss as our loss function since this was a multi-class classification problem. We used "SGD" as our optimizer since it would escape the saddle points in the training and it worked especially well for the optimization of such deep neural nets for which the landscape would be quite complex in terms of convex functions. We set a learning rate scheduler which would decrease learning rate with a factor of 0.1 every 7 steps.

Fourth, we computed the accuracy on the validation set since the validation accuracy was used to compute the termination condition of the training process. We switched the model to evaluation mode, and passed the data sets to GPU for processing. Then, we used the model to compute the predictions on the validation data set batch by batch (with batch size of 24). After decoding the one-hot-encoded predictions, we can compute the accurate predictions on each batch. After iterating over all the batches, we collected the number of counts of corrected classified labels which can be used to compute the validation accuracy. Fifth, we computed the termination condition based on the recently computed validation accuracy. For our self-implemented model, we collected every 100 validation accuracy and compute the standard deviation on all of them. If the increase of the standard deviation of the later 100 validation accuracy was less than 0.2, we made the algorithm terminated and regarded it as converged. The other termination condition was the maximum number of epochs the training process need, which was mentioned above.

Fifth, we make one step forward on the scheduler. Then, we switched the model to training mode with all the gradients in the optimizer being zero-initialized. By computing the loss on every batch the algorithm iterating through, we could use the back-forward method to do gradient descent on the neural network, which was the weight-optimization part. Meanwhile, we also collected the correct counts of prediction on the training set to compute the training accuracy.

Sixth, we used the accuracy gained from the validation and training data sets to draw the graphs below, and showed the trend of accuracy changing on these two data sets. The steps are shown in Fig.5.

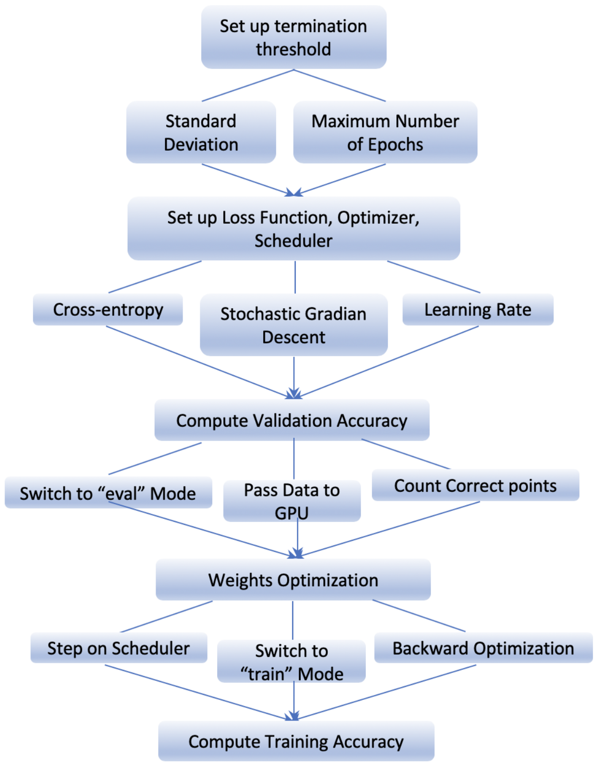


Figure 5: Training steps for training process

### 4.2 Training Results

For the model we imported from the library, we gained a final training accuracy of 75% and a validation accuracy of 85% The algorithm terminated at epoch 4 (start from epoch 0). The change of training accuracy and loss is shown in Fig.6 and Fig.7

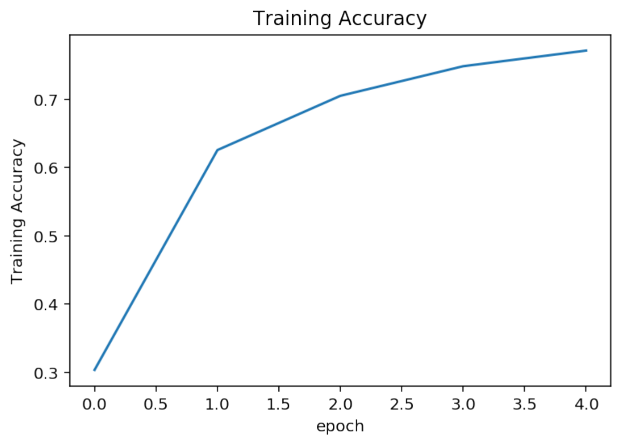


Figure 6: Training accuracy for training process with the model from library

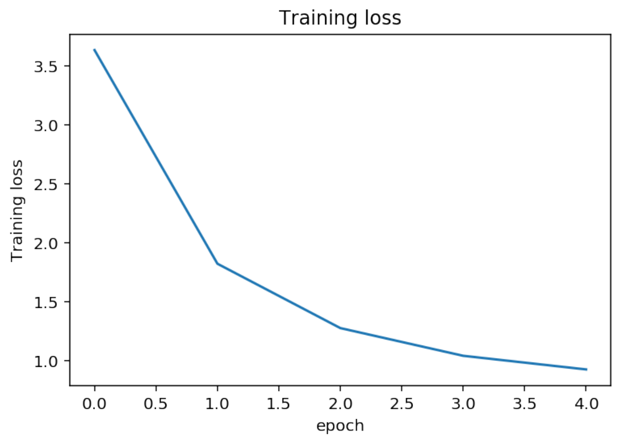


Figure 7: Training loss for training process with the model from library

For the model we built ourselves, we gained a final training accuracy of 6% , and a validation accuracy of 6% The algorithm terminated at epoch 83 since the GPU memory is saturated.The change of training accuracy and loss is shown in Fig.8 and Fig.9

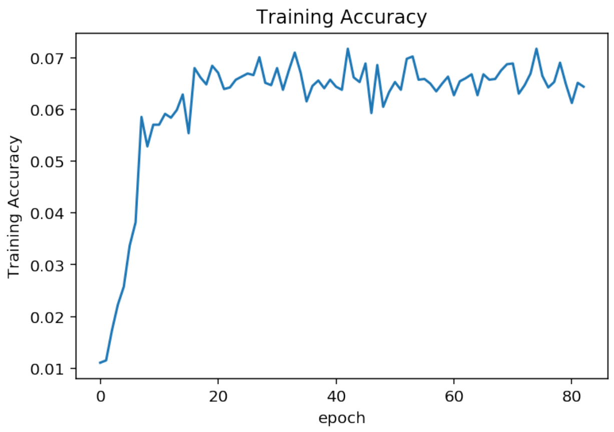


Figure 8: Training accuracy for training process with the self-implemented model

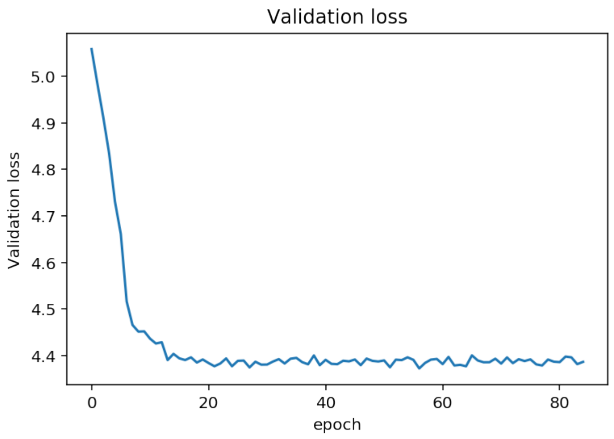


Figure 9: Training loss for training process with the self-implemented model

### 4.3 Comparison of library-imported and self-implemented model

From the result above, we can easily find out that transfer learning is crucial in neural network training process. For the model we imported from the library which was embedded with pre-trained weights, we can train the model in only 5 epochs and consider it as converged. It also can gain a very high training and validation accuracy that are 75% and 85%. However, for the model we implemented by ourselves which was not embedded with pre-trained weights, we can only get the training and validation accuracy at around 6%. Instead of increasing the accuracy on training and validation set significantly, the training process also used up all the memory of the GPU on Google Colab (14G). It turned out that without transfer learning, training the ResNet neural network was a heavy work. The training process was not only time-consuming, but also highly demanding for computation resource. From a pragmatic view, transfer learning is always a good choice for some well-investigated nerual networks. For example, the ResNet we used.

## 5 Evaluation

### 5.1 Transfer Learning Evaluation

The ResNet50 model used on Transfer Learning was evaluated on the validation set while training. The validation data set contained 845 samples which belonged to all 133 classes. The validation accuracy was recorded for every training epoch, thus a graph was plotted regarding the change of the accuracy with respect to training time. As shown on figure 9, the validation accuracy increased dramatically in the first 4 epochs. In particular, after the first iteration of training, the accuracy takes a huge leap from 0.007 to 0.65. At the end of the training, the accuracy was converged to 0.85.

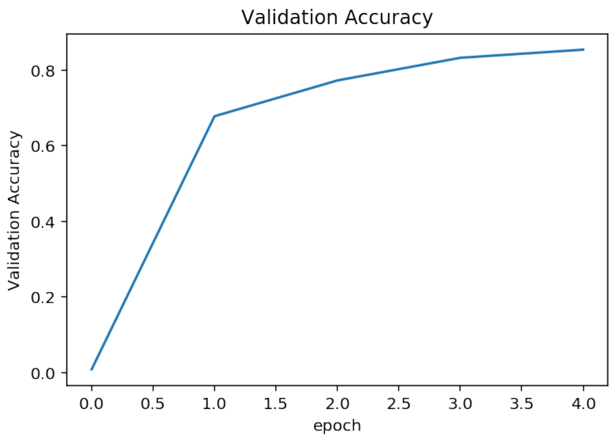


Figure 10: Validation accuracy for built-in ResNet50 training process

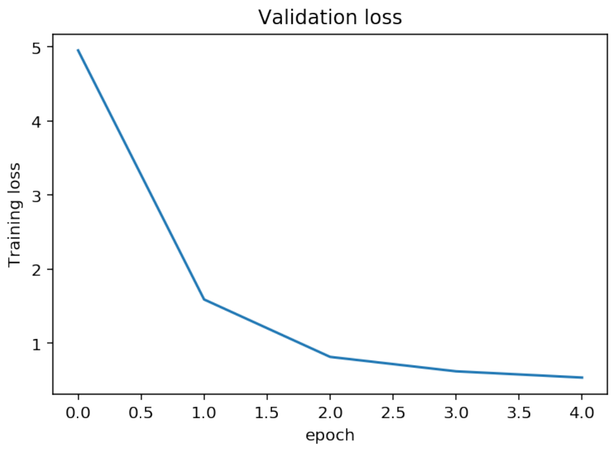


Figure 11: Validation loss for built-in ResNet50 training process

Once the training course wasn finished, the model was evaluated on a test data set which it had never been trained on. The testing data set was constructed by PyTorch Dataloader package. The testing set contained 836 images. All the samples were shuffled whenever Dataloader was called, and then all the random samples were fed into the model for final test evaluation under the form of 35 small batches. Each small batch contained 24 testing samples with ground truth label attached. At each round, the model would give predictions for 24 images in a batch and return a 1x24 vector. The returned vector would be compared with the ground truth labels inside the Dataloader which was also a 1x24 vector. The comparison between two vectors would result in a binary vector which contained an one at index j if the image corresponding to jth element in the ground truth label vector was correctly predicted, otherwise a zero. All the binary vectors would be summed up into a scalar value after 35 iterations, and the test accuracy can be computed by summation of binary ones / len(dataset['test']). The final test accuracy calculated was 89.7 %. The following figure showed that the top 30 breeds that were classified wrongly in 836 samples.

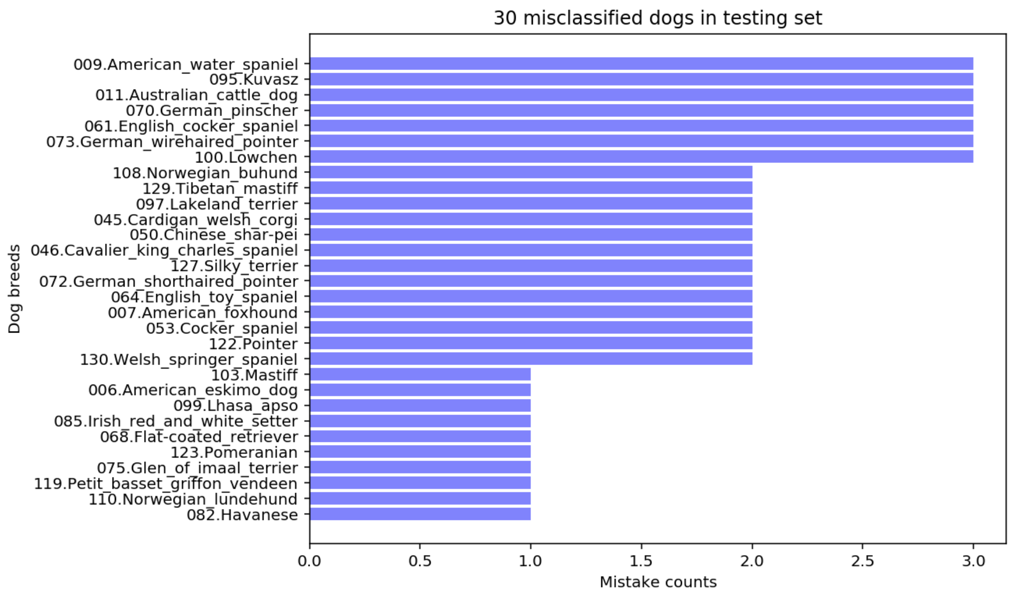


Figure 12: first 30 mis-classified dogs

The model was performing well within a reasonable error range. If we looked into an example shown on the above mistake histogram, we would understand why the model would make such mistakes.
Below are two dog images for Kuvasz(left) and Labrador(right):

As the pictures shown above, these two dogs come from different breeds but looks extremely similar. The only distinguishable feature is the hair length. However, given the background color and minute hair length difference. The algorithm certainly had chance to make mistakes here.

### 5.2 Self-implemented ResNet50 Evaluation

The training course became quite computational expensive for a self-implemented ResNet50 from scratch. The same validation set was applied on the model while training. The evaluation did not have much meaning here because it grew extremely slowly. The validation accuracy and validation loss were separately plotted in a relative small scale,because of the tiny change, the trend of both graphs could be treated as straight lines in a large scale after running close to 100 epochs. The accuracy was insignificant compared with what we achieved in transfer learning. The computation power we possessed did not support us to fully train a usable Res50 for image classification. Therefore, we did not evaluate its performance on the test set.

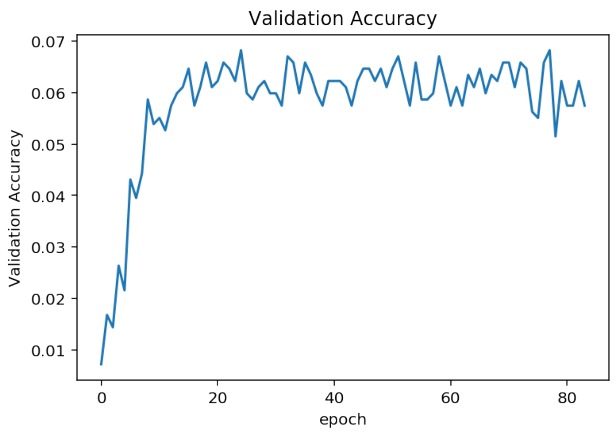


Figure 13: Validation accuracy for self-implemented model

## 6 Prediction

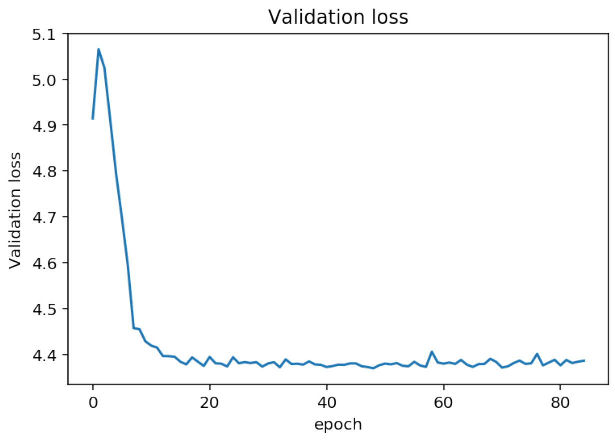
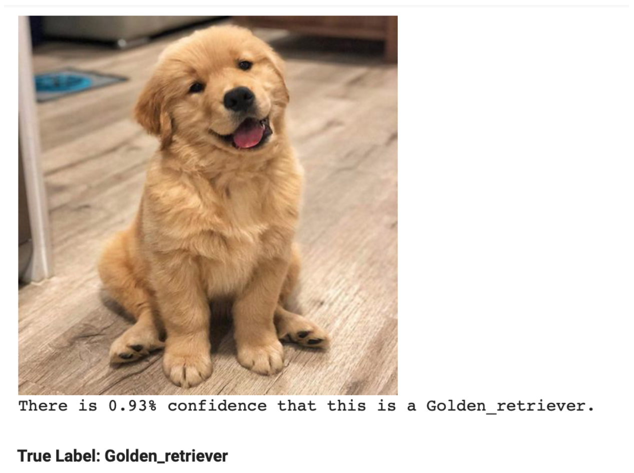


Figure 14: Validation loss for self-implemented model

## 7 Prediction

We make a prediction API for any input dog images to predict their breeds. In the prediction function, only a path containing the image is needed to pass into that function. Then the function reads the image in and convert it into a PyTorch tensor using data transformation method. The image will be resized and croped in the exact same way as the images input for training. Then the tensor will be passed into the model and the model returns a list of scores for each of the classes. The scores will further go into a softmax layer we built externally. The softmax will turns each score into a confidence about each class. And we extract the highest confidence and its index in order to looking for the class name. In the end, the input image would be shown as a part of the result along with one prediction statement.The sample output is shown in figure 14.



```
There is 0.93% confidence that this is a Golden_retriever.
```

**True Label: Golden_retriever**

Figure 15: sample output

## 8 Conclusion

In this project, we implemented the ResNet50 neural network on a classification problem which is to classify the breeds of dogs. In the training process, we find out that transfer learning is important for training of neural network. By using transfer learning, we can easily gain the accuracy on training and validation data sets at 75% and 85% when iterating on epochs. When training without transfer learning, we can only get the accuracy both on training and validation data sets at around 6% when iterated 83 epochs. It is easy to see that without transfer learning, the training process is not only time-consuming, but also highly demanding for computation resource. Overall speaking, the classification work for dog breeds was successfully achieved by using the built-in model and pretrained weight in PyTorch. The testing accuracy is close to 90 % only under the training for around 10 epochs, which essentially took up 8 - 10 minutes.In the end, we also developed a function which allowed user to input any dog images to recognize. On the other hand, implementing a ResNet50 convolutional neural network by ourselves encouraged us to dive into the detailed architecture and helped us understand mathematical theory of it: with a extra identity added to the block output, we manually help to increase the derivative of weights in some certain layers and eventually prevent gradient vanishing for deep neural nets. Besides understanding the ResNet, implementing one required us to study and utilize deep learning framwork like PyTorch which is crucial for machine learning students.

## 9 Limitations and Future Works

### 9.1 Limitations

Due to various reasons, two major limitations are presented in this project. First, the dataset contains only 8361 images but 133 categories of dogs. Therefore, it is difficult to build a comprehensive and accurate model given this amount of data. Second, the applied data augmentation methods might not improve the result significantly as the augmented data shares lots of similarities with the original data. As we focus more on building the model for the project, it is reasonable that we are unable to spend much effort on data processing given the time constraints.

Second, due to the limited computing resource, training process terminated before reaching our expectation when we implemented our self-built model. As a result, we could only project expected results from current result, causing uncertainty and inaccuracy in general.

### 9.2 Future Works

In general, the future works should majorly involve in exploring CNN for dog breed prediction. Despite having some limitations, the project itself has a great potential for expansion. First, merging current dataset with more datasets could potentially increase accuracy of training/prediction results. Second, further exploring transfer learning could significantly improve the efficiency of the training process (as discussed in previous sections). Third, besides ResNet50, other models could be explored and compared with more available computing resources. Training a convolutional neural network is always time-consuming, so more efforts need to be put into parameters adjustment and model building in order to guarantee efficiency while not losing accuracy.

## References

[1] A. H. Freedman, R. M. Schweizer, I. Gronau, E. Han, D. O. D. Vecchyo, P. M. Silva, M. Galaverni, Z. Fan, P. Marx, B. Lorente-Galdos, et al. Genome sequencing highlights genes under selection and the dynamic early history of dogs. In *arXiv preprint arXiv: 1305.7390* , 2013.

[2] George Kour and Raid Saabne. Fast classification of handwritten on-line arabic characters. In *Soft Computing and Pattern Recognition (SoCPaR), 2014 6th International Conference of*, pages 312–318. IEEE, 2014.

[3] Guy Hadash, Einat Kermany, Boaz Carmeli, Ofer Lavi, George Kour, and Alon Jacovi. Estimate and replace: A novel approach to integrating deep neural networks with existing applications. *arXiv preprint arXiv:1804.09028*, 2018.