Using Convolutional Neural Networks to Classify Dog Breeds

Hsu, David
Stanford University

fcdh@stanford.edu

Abstract

Dog breed categorization is a very specific application of convolutional neural networks. It falls under the category of fine-grained image classification problem, where inter-class variations are small and often one small part of the image considered makes the difference in the classification. The various classes of ImageNet can have large inter-class variations, making it easier to categorize correctly. In this work, I aim to use a convolutional neural network framework to train and categorize dog breeds. I approach this first using CNNs based on LeNet and GoogLeNet architectures.

1. Introduction

Convolutional neural networks (CNN) have been used to great effect in applications such as object classification, scene recognition, and other applications. In many situations, we can imagine the features (both low-level and higher-level) that are learned by the CNNs in the process of training. However, when the objects the CNNs are trying to categorize share many similar features, such as the breeds of dogs, it becomes hard to imagine the specific features that CNNs must learn in order to categorize these dogs correctly. This is especially true if we take a look at sets of images such as Fig. 1 below, where the 3 dogs share almost all the same visible features, but belong to different classes. It is therefore interesting to see how well CNNs can perform on only dog breeds, compared to labels from all classes of objects in the regular ImageNet.

Fig. 1a: Malamute

Fig. 1b: Eskimo Dog

Fig. 1c: Husky

1.1. Background

The problem of fine-grained image classification (FGIC) has led to the creation of many FGIC datasets that span a wide variety of categories, including Stanford Dogs [1], CalTech-UCSD Birds [2], Oxford Flowers [3], FGVC-Aircrafts [4], and more. We can see that the types of objects in these datasets vary, from being rigid to deformable, and variant or invariant in colors and shape across the same class.

Different approaches have been explored for FGIC. A traditional method is to use different descriptor extraction algorithms, and to run a linear classifier on the features that are extracted. Khosla et. al, who created the Stanford Dogs dataset, were able to achieve 22% accuracy using SIFT descriptors for classification on Stanford Dogs[1]. The current method for achieving the highest accuracy (52%) on the Stanford Dogs dataset is by using Selective Pooling Vectors [5], which encodes descriptors into vectors, and selects only those that are below a certain threshold of quantization error, with respect to the codebook which is used to approximate the nonlinear function f used to determine the classification likelihoods of various classes.

Another approach is to "localize" various landmarks inside the particular class, and to co-register these landmarks and perform comparisons on them. Both supervised and unsupervised learning methods have been applied here. Again, feature extraction methods (such as SIFT) can be used to localize dog faces before classification is performed, as shown in [6]. Unsupervised learning methods have been developed to learn "template" shape patterns which commonly re-occur in all images being categorized, and have been able to show up to 38% accuracy on the Stanford Dogs dataset [7]. One paper [8] utilizes R-CNNs to categorize the CalTech-UCSD Birds, by leveraging CNNs in both the part localization and image classification portions of the tasks. This was extended to two-level attention models [9], where the images are split into patches, with one level determining the relevance of the patch, and the next performing the actual classification. Approaches involving gnostic fields [10] have achieved some of the highest levels of accuracy on the Stanford Dogs dataset (47% accuracy), using pattern-detection units and image descriptors to create a size and shape-invariant model

which can consider differently-sized images without suffering from biases introduced by differences in image size.

The table below summarizes the current benchmarks on the Stanford Dogs dataset that I discussed above.

Table 1: Summary of Benchmarks of Stanford Dogs

Method [citation]	Top - 1 Accuracy (%)
SIFT + Gaussian Kernel [1]	22%
Unsupervised Learning Template [7]	38%
Gnostic Fields [10]	47%
Selective Pooling Vectors [5]	52%

1.2. Approach

My main motivation for this class project was twofold: to learn about Caffe and how to set up/train CNN's on Caffe, and to use an image dataset that involves dogs because I like dogs. The short timespan of the class meant that my project was limited in scope, and I did not have time to try many approaches that I would like to, which will be talked about in the conclusion section.

I settled on trying various regular CNN architectures, including LeNet [11] and GoogLeNet [12], in this task. I wanted to evaluate the performance of the two different network architectures. I decided to use LeNets and GoogLeNets for several reasons. The first reason is that the LeNet architecture, with CONV-RELU-POOLs stacked with fully connected layers at the end, have been shown to perform well on many classification problems such as ImageNet, notably shown by Krizhevsky et. al. in 2012 [13]. I chose to try out GoogLeNets because of its novel architecture, its ease of training, and the lower number of parameters that it uses. According to Google's paper, bigger networks should come with increases with performance, but are more prone to overfitting. Since I am dealing with training with a comparatively small dataset, it makes sense to choose an architecture which reduces overfitting but maintains good performance. GoogLeNet provides this, and also it has not been tried with dog classification before, all of which are reasons why I decided on this approach.

The LeNet architecture includes the following:
(CONV#-RELU-POOL) x N + (FC#) x M + FC-120.
In this representation, each CONV layer consists of filters with two parameters, size and depth. Size refers to the number of pixels the filter spans, while the depth refers to the number of channels that are at the output of the CONV layer and fed into the next layer.

The GoogLeNet architecture includes "inception layers" shown in Fig. 2, which are a combination of network-in-network filters, along with convolutional filters of different sizes per layer. Each network-in-network filter is followed by either 3x3 or 5x5 filters, before they are all combined (using a concatenate function in Caffe) and pooling is performed. In this way, there are multiple filter sizes per layer so each layer of image sizes are given the ability to target the different "feature resolutions" that may occur in the input of that layer.

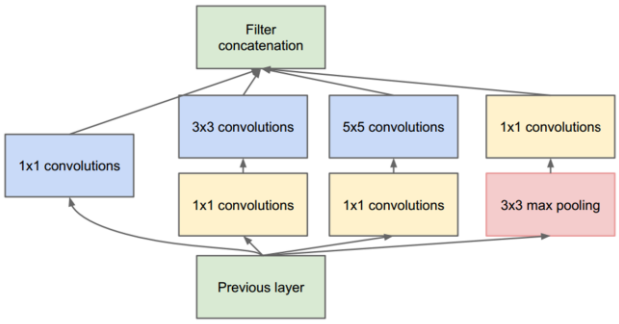

Figure 2: Construction of One Inception Layer
(Note, image is directly sourced from [12])

1.3. Experiment

The Stanford Dogs dataset is an open-access image dataset of dog breeds. There are a total of 120 classes of dogs, with 20580 images in total, partitioned into 8580 test images, and 12000 training images. The image dataset comes with annotations that mark out the bounding boxes which best include the dogs in the image. The dimensions of both the bounding boxes and the original images vary across the board, and the scenes are non-uniform in style within one single class, with occlusion, different poses, different background objects, different colors of fur.

The first step in the learning process is to create a usable set of images for training and testing from the raw image files. The first thing that I did is to crop all images using the annotated bounding boxes. The next step is to resize all resulting images to 256x256 for training and testing purposes. For this, I decided against simply brute-force resizing it because I didn't think that filters would be able to evaluate a squished or stretched image in the same fashion. Therefore, I decided to throw out all cropped images which had one of the two dimensions under 256 pixels, and for the remaining images, I would resize both dimensions down equally until the smaller dimension was 256, before taking rows and columns [1:256] from the image. I realize that repeated random sampling of rows and columns from the image can create an augmented dataset which should improve training, but for this project I did not do that. After this preprocessing, I ended up with 5678 training images, and 4007 testing images, which I converted into LMDB

format for use with Caffe. The overall image resizing steps are shown in Fig. 3 below.

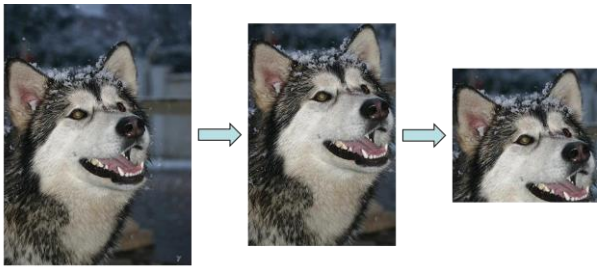

Fig. 3: Image Resizing Steps Taken

The learning framework/software that I decided to use was Caffe. There were many reasons for this. First, the class encourages students to learn Caffe, which is an open-use software package that has an optimized tradeoff between ease of use and speed. Also, porting the training from CPU to GPU is very easy, as it only involves setting a flag. The prototxt format is also quite easy to understand and make work. However, I had many issues getting Caffe to work for my project. I started off installing Caffe in Ubuntu on Virtualbox, and running the training on the CPU in the virtual machine. However, I abandoned the effort soon after I realized that this would not go anywhere, as the machine would be able to do only 500 or so iterations over 12 hours. The training went much faster after I moved to Terminal and used their GPUs to train, but I had lost a large chunk of time in the process.

All of the results from the testing of the nets are summarized in Table 2. The testing accuracy are based on the cropped and resized testing dataset, using Top-1 accuracy. All learning was done using learning rate of 0.0005, momentum of 0.9, weight_decay of 0.0005. All CONV layers were regularized with the weight_decay. The hyperparameters were determined through 10+ fine-tuning iterations using different parameters from the parameter space to determine the optimal one for the problem as I have defined it.

I first started testing with LeNets, going from 3 layers of CONV-RELU-POOL, all the way to 6 layers. I experimented with varying the convolutional filter depths (50 and 500) along the way, while sticking to a filter size of 5x5. The accuracy results I obtained after 100,000 iterations were not good (< 2%) when I have less than 6 layers with 1 fully-connected layer, and when I increased the network to be 6 layers with 2 fully-connected layers, it went up to 9.4% accuracy.

For my testing with GoogLeNets, I used a similar approach with the number of layers, going from 3-7 layers. I kept the size of the filters per layer the same as in Fig. 2, and only varied the depths of the filters in my testing.

One thing I noticed early on with training is that the quality of convergence of the nets vary wildly depending on the learning rate. For example, if I use a learning rate of 0.0001, even at 100,000 iterations, the accuracy of the model will still be less than 2%, while if I use a learning rate of 0.0005, the model converges quite well to the maximum accuracy, and a learning rate of 0.001 causes the loss to blow up within 2000 iterations.

I visualized some filters below from one example of LeNet that I trained. Unfortunately I experienced a weird bug in pycaffe when I tried to visualize some of the larger LeNets or GoogLeNets. In these situations, if I load the model with pycaffe, all the weights would appear to be NaN's and the test accuracies would appear to be uniformly distributed across all classes, but when I use the caffe –test interface in terminal, it would give an accuracy with a low testing loss (~0.5) as well as the accuracy it displayed while testing (~9%). I am not sure why this would be the case, and I wonder if my computer has memory issues which make it unable to load larger nets. Therefore, the visualization below is only from a 3-layer LeNet, with depth-50 filters, which achieved below 2% accuracy during testing. Although I would have preferred a visualization of a CNN that did better, I figured that it was better to have some visualization than none at all.

The visualizations below show the original image (Fig. 4) that was fed into the filters, the first and second layer filters themselves, and outputs from the first and second convolutional layers. The first layer filters (Fig. 5) are low enough in number, with small number of channels (3 channels for RGB since they are the first layer), that we can visualize them by simply combining the 3 channels and representing them in RGB format. Therefore, the first layer filters are shown in color, and there are 50 of them, representing a depth of 50. The outputs from the first layer Fig. 6 are plotted in grayscale, and show the result of convolving the image below with the filters. To make it convenient for viewing, I am only showing the outputs from the first 36 channels, so the resulting image matrix is square. We can already see some interesting results from the output of the first layer. Firstly, in some of the outputs, the contrast between the white and black portions of the dog are maintained (though they may be reversed so that black becomes white and vice versa), but in other outputs there are barely any differences between the two. We can see that in the 50 inputs to the second layer for this particular image, some inputs will have barely any signal and contrast, while others will maintain their strong contrasts for edge detections and other non-linear operations.

The second layer filters, while visualized in Fig. 7, are not really able to be viewed in an easy manner. This stems from the fact that there are 50 input channels per filter, and there are 50 filters in the layer as well. The Caffe tutorial suggests that these filters be plotted in a row-column matrix format,

where every row indicates all of the input channels of the filter, and the column represents the depth of the filters.

I do note here that the outputs from the second layer (Fig. 8) are not too different from the first layer. This is a good representation of a network that is not well-learned, because there is not much transition of hierarchy of low-level to high-level features as we move through the levels of the network.

Fig. 4: Original Image for Visualization

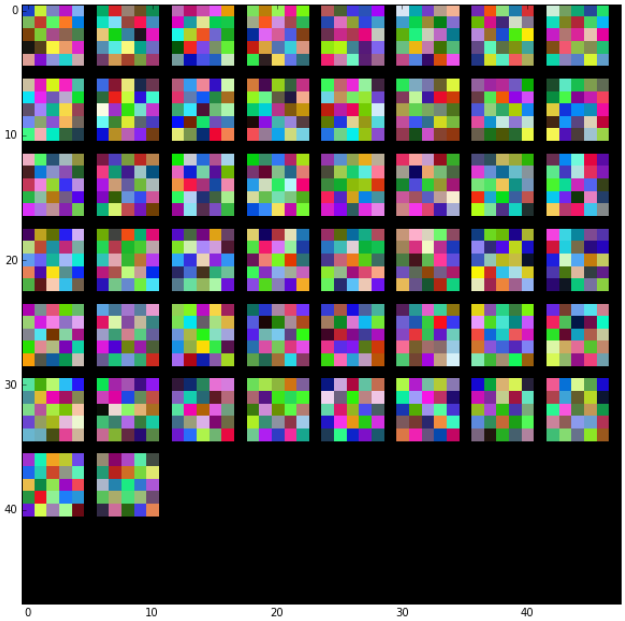

Fig. 5: Visualizing First Layer CONV Filters

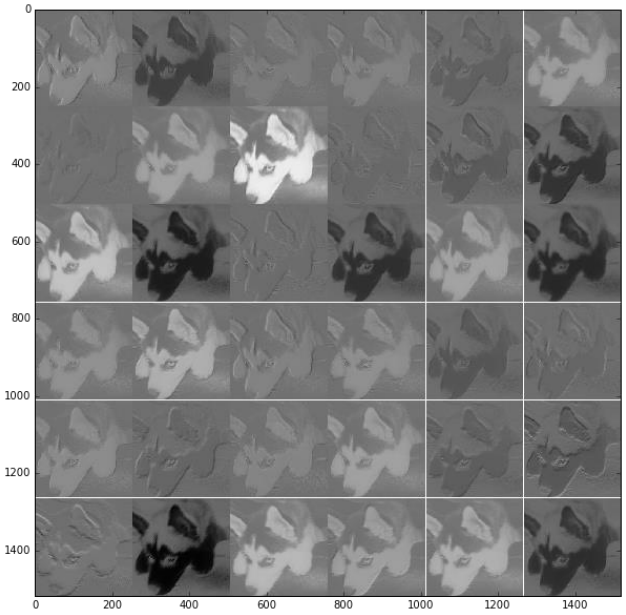

Fig. 6: Output from First CONV Layer

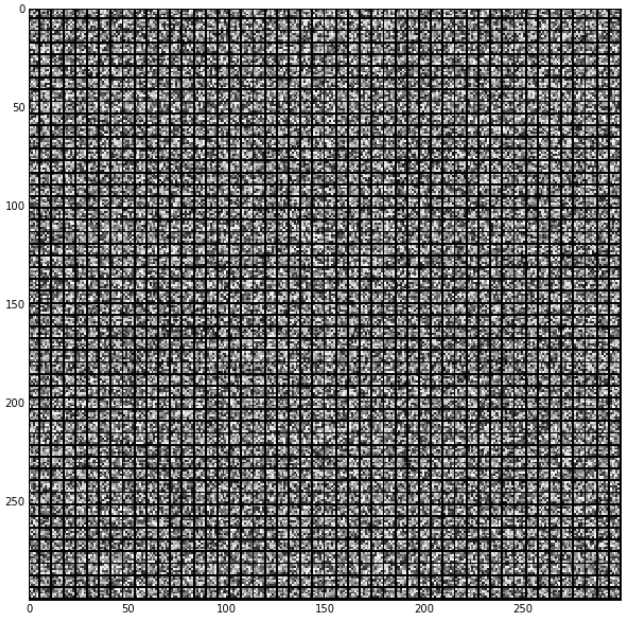

Fig. 7: Visualizing the Second CONV Layer

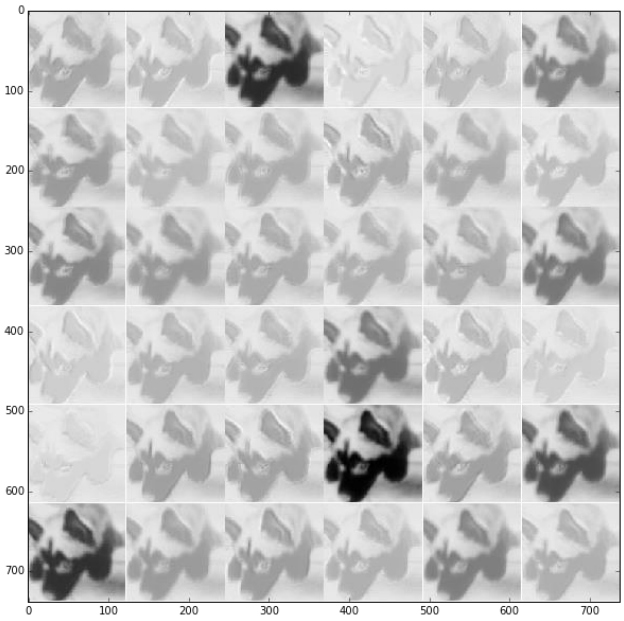

Fig. 8: Output from the Second CONV Layer

Notes for the table below
- (CONV-RELU-POOL# is abbreviated CRP#), with the # representing the depth of filters (all LeNet filters tried are 5x5)

- All GoogLeNet layers contained 2 FC-500 layers and a FC-120 layer at the end

Table 2: Results of CNN Testing

Method	**Top-1 Accuracy (%)**
Random Guessing	0.83%
LeNet: (CRP-50 x 3)+(FC-1000)+(FC-120)	< 2%
LeNet: (CRP-500 x 5)+(FC-1000)+(FC-120)	< 2%
GoogLeNet: 3 layers, 16 filters per CONV	< 2%
GoogLeNet: 4 layers, 16 filters per CONV	3.4%
GoogLeNet:7 layers, filter 16x2+64x2+128x3	5%
GoogLeNet:6 layers, filter 16x2+64x2+128x2	8.9%
LeNet: (CRP-500x6)+(FC-1000)x2+(FC-120)	9.5%

1.4. Conclusion

There are many things that I have learned from doing this project. Firstly, I have learned how to utilize Caffe to run categorization with CNN's. To me, this is the most important takeaway from this class, because I wanted to apply the concepts of neural nets to my research in the Radiology department in the medical school (which is the reason why I decided to take this class in the first place). Secondly, I learned that it is not easy to optimize the hyperparameters for CNN training, since it took a few days before I could control the learning rate so that the accuracy would go up without blowing the loss up to NaN. Thirdly, I learned from an agonizing week-long trial with running Caffe on CPUs that it is better to set up Caffe on a machine with access to GPUs. The comparison between the two was staggering, with the CPU machine taking 2 days to train about 1000 iterations, and the GPU machine taking about 8 hours to run 100,000 iterations.

There are lots of future work that can be done on this project. To start with the basics, if I want to run the training of CNNs in this manner, the least I should have done is to initialize some of the layers with pre-existing AlexNet or GoogLeNet models that are pre-trained on the ImageNet classification. The uncertainty here is how the CONV filters in these pre-trained models, which respond to features present in non-dog images, would respond to the training of dog images. The second step would be to perform transfer learning, by chopping off the final FC layers from the ImageNet models, and using the CNN codes to perform regular SVM or Softmax classification. More complex work would include incorporating methods of part localization into the classification flow, so faces/eyes/ears can be considered with different filters and provide a better accuracy.

I would like to thank the teaching staff for the execution of this class. I realize this project is not well-executed, but that is solely because of the lack of time due to my busy research schedule for the past weeks, and not a reflection on the way the class is structured. I learned a lot and genuinely had a great time in this class. Thank you again.

References

[1] A. Khosla, N. Jayadevaprakash, B. Yao and L. Fei-Fei. **"Novel dataset for Fine-Grained Image Categorization".** *First Workshop on Fine-Grained Visual Categorization (FGVC), IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2011.

[2] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. **"The Caltech-UCSD Birds-200-2011 Dataset."** *Computation & Neural Systems Technical Report,* CNS-TR-2011-001.

[3] M. Nilsback, and A. Zisserman. **"Automated flower classification over a large number of classes."** *Computer Vision, Graphics & Image Processing, 2008. ICVGIP'08. Sixth Indian Conference on*. IEEE, 2008.

[4] S. Maji, J. Kannala, E. Rahtu, M. Blaschko, A. Vedaldi, **"Fine-Grained Visual Classification of Aircraft",** *arXiv.org*, 2013

[5] G. Chen, J. Yang, H. Jin, E. Shechtman, J. Brandt, and T. Han, **"Selective Pooling Vector for Fine-Grained Recognition"**, *Applications of Computer Vision (WACV), 2015 IEEE Winter Conference on*. IEEE, 2015.

[6] J. Liu, A. Kanazawa, D. Jacobs, and P. Belhumeur, **"Dog Breed Classification Using Part Localization"**, *Computer Vision–ECCV 2012*. Springer Berlin Heidelberg, 2012. 172-185.

[7] S. Yang, L. Bo, J. Wang, and L. Shapiro, **"Unsupervised Template Learning for Fine-Grained Object Recognition".** *Advances in Neural Information Processing Systems*. 2012.

[8] N. Zhang, J. Donahue, R. Girshick, and T. Darrell, **"Part-based R-CNNs for fine-grained category detection".** *Computer Vision–ECCV 2014*. Springer International Publishing, 2014. 834-849.

[9] T. Xiao, Y. Xu, K. Yang, J. Zhang, Y. Peng, and Z. Zhang, **"The Application of Two Level Attention Models in Deep Convolutional Neural Network for Fine-Grained Image Classification"**, *arXiv.org:1411.6447*, Nov. 2014.

[10] C. Kanan, **"Fine-Grained Object Recognition with Gnostic Fields"**, *Applications of Computer Vision (WACV), 2014 IEEE Winter Conference on*. IEEE, 2014.

[11] Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, P. Simard, and V. Vapnik, **"Comparison of learning algorithms for handwritten digit recognition**." *International conference on artificial neural networks*. Vol. 60. 1995.

[12] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, **"Going Deeper with Convolutions",** *arXiv:1409.4842*, Sept. 2014.

[13] A. Krizhevsky, I. Sutskever, and G. Hinton**. "Imagenet classification with deep convolutional neural networks"**, *Advances in neural information processing systems*. 2012.