CS230

# Fine-grained Image Classification

**Debabrata Sengupta**
dsgupta@stanford.edu

## Abstract

My goal was to implement a classifier which is able to perform fine grained image classification to distinguish between different breeds of dogs, using the Stanford Dogs dataset. I tried several approaches, starting with my own small CNN architecture, a VGG16 network and also a ResNet50. I settled on VGG16 with transfer learning. Accuracy of the network was improved through regularization and data augmentation. I achieved a classification accuracy of about 41%, which while far below state of the art, is significantly better than what the authors of the dataset first achieved. Error analysis was also performed to gain insights into model performance.

## 1 Introduction

While image classification involves trying to discriminate between different classes of objects, fine-grained image classification usually involves discriminating between sub-classes of an object. This class of problems is more challenging than image classification because it is characterized by large intra-class variation and small inter-class variation.

In the Stanford Dogs dataset, each class is a breed of dogs. Dogs within the same breed can often have large variation in terms of color (e.g. Shih-tzu). On the other hand, two breeds of dogs can share same facial characteristics (e.g. basset hound and blood hound). Puppies of one breed can look like dogs of another breed due to their small size. Furthermore, images in this dataset often have occlusion and difficult backgrounds since they include human beings and cluttered environments.

While several computer vision applications make use of specialized feature detectors (like SIFT, HOG, etc.), another motivation for me was to see whether I could bypass hand-crafting feature detectors completely and let a convolutional neural network learn distinguishing features by itself. The input to the CNN would be the raw image (after minor pre-processing steps) and the network would learn to directly output the breed of the dog in the image.

## 2 Related work

There are several common datasets that papers on fine-grained image classification use to benchmark their results. Stanford Dogs, CalTech-UCSD Birds, Oxford Flowers, etc. are all quite popular.

One traditional approach for fine-grained image classification is to use descriptor extraction algorithms and then run a classifier on the features that are extracted. The authors of the Stanford Dogs dataset used SIFT descriptors to achieve 22% accuracy on this dataset [1]. Authors of [3] achieve 52% accuracy by using Selective Pooling vectors. Ensemble approaches were used in [2], where the authors trained K expert classifiers and then aggregated their results. The current state of the art for Stanford Dogs [4] achieves 88.9% accuracy using fully convolutional attention networks.

,

Figure 1: Representative images from the dataset. Top, first pair shows different fur color in Shih-tzu, second pair shows similar facial characteristics in basset hound and blood hound. Bottom shows background clutter. Images are of different sizes.

## 3 Dataset and Pre-processing

For all my experiments, I used the Stanford Dogs dataset [1]. This dataset contains over 22,000 annotated images of dogs belonging to 120 species. Each image is annotated with a bounding box containing the dog, and a class label. On average, there are between 150 and 180 images in each class. The dimensions of images vary, and so do the dimensions of the bounding box. Furthermore, the images within a single class do not have uniformity in terms of style, with occlusion, background clutter, variation in poses, variation in fur color all complicating the dataset. The dataset is partitioned into 12000 training images and 8580 test images.

The first step in my project was to pre-process this dataset and create my train, validation and test set. Since images have varying dimensions, I decided to first crop each image to extract only the bounding box around the dog and then resized these cropped images to size 224 x 224 x 3. The reason for choosing this size was that I wanted to use a VGG16 like CNN as my neural network architecture, and that uses image inputs of this size. After resizing, I randomly split the training set 80:20 to create the train and validation set. The dataset already provides the test set, so I didn't need to carve out the test set separately.

Following this, the next step was normalizing the input images to have values in [0,1]. One-hot encoding was performed on the class labels to get a 120 dimensional vector. This completed the pre-processing step.

## 4 Methods

### 4.1 Training a CNN from scratch

Although having only about 80 images per class is very small for training and the model would probably overfit, I still decided to train a small CNN from scratch using my own VGG16-like architecture. I did not use VGG16 (or any other classic network) because of two reasons – first, I wanted to see how severe the overfitting problem is, and second, I wanted to use a simpler network than VGG16 that would train faster on a CPU and overfit less to the training data.

The architecture that I used is shown in Fig 2. I used Keras for implementing my CNN. There was a Batch normalization step before every convolution block. I used He initialization for all parameters. The objective function being optimized was minimizing the cross entropy loss. There was no regularization applied in this first model. I used Adam optimization with a mini batch size of 32. Along with the loss at each epoch, the model also output the training and validation accuracy. I check-pointed the model after every epoch and retained the model with the best validation accuracy.

The results of this experiment confirmed what I had expected – the model was massively overfitting to the training set. After about 15 epochs, the training accuracy reached almost 90%, while the validation accuracy peaked at about 30%. This huge difference between training accuracy and validation accuracy showed that the model was overfitting. The test accuracy was close to the validation accuracy, 31.55%.

| Layer (type) | Output Shape |
|---|---|
| batch_normalization_13 (Batc | (None, 224, 224, 3) |
| conv2d_11 (Conv2D) | (None, 222, 222, 16) |
| max_pooling2d_11 (MaxPooling | (None, 111, 111, 16) |
| batch_normalization_14 (Batc | (None, 111, 111, 16) |
| conv2d_12 (Conv2D) | (None, 109, 109, 32) |
| max_pooling2d_12 (MaxPooling | (None, 54, 54, 32) |
| batch_normalization_15 (Batc | (None, 54, 54, 32) |
| conv2d_13 (Conv2D) | (None, 52, 52, 64) |
| max_pooling2d_13 (MaxPooling | (None, 26, 26, 64) |
| batch_normalization_16 (Batc | (None, 26, 26, 64) |
| conv2d_14 (Conv2D) | (None, 24, 24, 128) |
| max_pooling2d_14 (MaxPooling | (None, 12, 12, 128) |
| batch_normalization_17 (Batc | (None, 12, 12, 128) |
| conv2d_15 (Conv2D) | (None, 10, 10, 256) |
| max_pooling2d_15 (MaxPooling | (None, 5, 5, 256) |
| batch_normalization_18 (Batc | (None, 5, 5, 256) |
| global_average_pooling2d_3 ( | (None, 256) |
| dense_3 (Dense) | (None, 120) |

Figure 2: Layers of my own CNN.

### 4.2 Transfer Learning using VGG16 network

Clearly, the primary problem that my first model faced was overfitting. Few ways to fix this problem would be to use regularization, data augmentation, and use of transfer learning. Out of these three, regularization seemed the least promising approach to me because the gap between training and validation accuracy was as large as 60% and regularization alone would not help narrow such a large gap.

I first decided to first try out transfer learning along with data augmentation. Keras provides a pre-trained VGG16 model, which has been trained on ImageNet. Transfer learning works in this case because the Keras model was trained on a very large amount of data and the inputs in both cases are fundamentally similar. I decided to drop the top few layers of the Keras model and instead added in two fully connected layers and a softmax classification head of my own, with 120 classes. I froze all the weights of the pre-trained VGG16 network (after dropping the top layers) and used my own training set to train the weights of only my fully connected layers and softmax layer. Again, the optimization problem remained the same – minimizing the cross entropy loss without any regularization, using Adam optimization.

The results of this experiment were not very promising. While the model definitely converged faster because we had frozen a large part of the network and had fewer parameters to train, I observed that the validation accuracy did not improve much beyond 32%. Overfitting still remained a problem. The 'best' model (the one with highest validation accuracy) was reached in just 5 epochs, but training for more epochs only increased the train accuracy due to overfitting.

Next I tried using data augmentation with horizontal flips and shifts along horizontal and vertical axes by 15%. I also added in Dropout with keep_prob = 0.6 as a form of regularization. This approach worked very well. The gap between training and validation accuracy reduced dramatically. The best validation accuracy achieved was 40.2% and it was reached after 15 epochs.

### 4.3 Transfer Learning using ResNet50 network

Having trained a VGG16 model, I wanted to see if a much deeper network like ResNet50 would work if I used a combination of transfer learning, data augmentation and regularization. I used the ResNet50 model that is available in Keras, pre-trained on ImageNet. I froze all the layers and added in one Dense(512) layer and a 120 class softmax classification head. I also added in a Dropout layer with keep_prob = 0.6. This model performed very poorly, with extreme overfitting. With just 1 epoch, the training accuracy reached 41% while the validation accuracy was only 0.83%! Clearly, there was insufficient data to try out very deep networks and even data augmentation would not help. So I

decided to drop my ResNet50 approach and instead chose the VGG16 model as my final deliverable for this project.

## 5 Results and Discussion

As described above, I tried three different CNN architectures to perform this fine-grained image classification task. The classification accuracy that I obtained on my training, validation and test sets for each of the three approaches is summarized in Table 1.

| Table 1 : Classification Accuracy (%) | | | |
|---|---|---|---|
| Model | Training Set | Validation Set | Test Set |
| My own CNN architecture with regularization | 93.66 | 30.67 | 31.55 |
| VGG16 (Transfer Learning + Augmentation + Dropout) | 43.87 | 40.20 | 40.79 |
| ResNet50 (Transfer Learning + Augmentation + Dropout) | 94.14 | 23.83 | 24.72 |

As can be seen from the table, ResNet50 does the worst on the test set, primarily because it is a much deeper network and it overfits the most to the training data. VGG16 achieves a good balance in between the two approaches, helped by use of transfer learning, data augmentation and regularization.

Next, I spent some time doing error analysis. I plotted the confusion matrix (as a heat map) for the test set, shown in Figure 3. As expected, brightest points on the heat map are mostly along the diagonals, corresponding to correct classifications. There are however, several somewhat bright spots scattered among the off-diagonal elements, indicating that overall accuracy was not very high.

It is hard to read the confusion matrix for so many classes. So I also took the top 20 "Ground Truth - Prediction" pairs which have the most misclassifications to take a closer look. These pairs are shown in Figure 3. Investigating these more, we can see that in many cases the two classes are so similar that it is hard even for humans to distinguish between them. It is quite possible that the Bayes' error for classification among these particular breeds is quite high already. Some samples images for commonly misclassified breeds is shown in Figure 4.

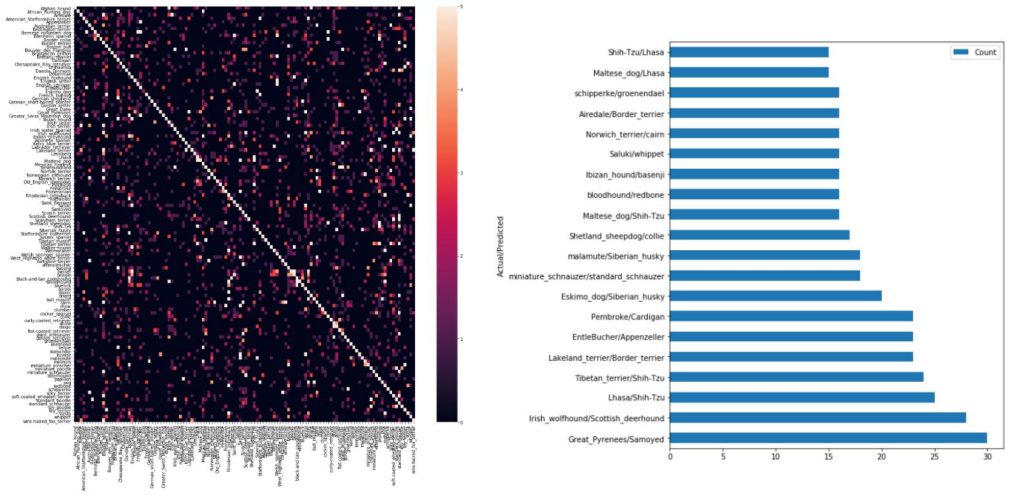


Figure 3: Left: Confusion matrix Heatmap. Right: Top 20 Actual/Predicted pairs by error.

(a) Great_Pyrenees (b) Samoyed

Figure 4: Sample images from the top pair which was misclassified. Images of dogs from both these breeds appear very similar even to humans.

## 6 Conclusion

In this project I went through the full cycle of pre-processing data, training a model, iterating over it to try out different ideas and performing error analysis, all while building a classifier for dog breeds. Given the scarcity of training data per category, transfer learning and data augmentation proved extremely useful. Shallower networks like VGG16 performed better than deeper networks like ResNet50 due to overfitting. While the best accuracy that I achieved is much lower than the state of art on this dataset, it is not too bad either given how challenging this dataset is. In fact, my model performs significantly better than the SIFT descriptor approach proposed by the authors of the Stanford Dogs dataset. These results prove that CNNs perform reasonably well without the need of hand-crafted features.

If I had more time to work on this project, I would have liked to explore ensemble approaches like learning K expert classifiers and aggregating their decisions.

## Github Repository

All my code is available at : `https://github.com/debtsetgo/cs230_dsgupta_final.git`

## References


[1] A. Khosla, N. Jayadevaprakash, B. Yao, F. Li, *Novel dataset for Fine-Grained Image Classification*.

[2] ZongYuan Ge, Alex Bewley, Christopher McCool, Ben Upcroft, Peter Corke, Conrad Sanderson *Fine-Grained Classification via Mixture of Deep Convolutional Neural Networks*.

[3] G. Chen, J. Yang, H. Jin, E. Shechtman, J. Brandt, and T. Han, *Selective Pooling Vector for Fine-Grained Recognition*.

[4] Xiao Liu, Tian Xia, Jiang Wang, Yi Yang, Feng Zhou and Yuanqing Lin, *Fully Convolutional Attention Networks for Fine-Grained Recognition*.