CS 280 MiniPlaces HW2 Part 3

Madeleine Traverse Kaggle: Dodecaninjasaur

March 13, 2017

1 Design

1.1 Improvements on Alexnet

The first model I implemented was a small modification to the Alexnet architecture. I based it on the "ZF Net" (2013)[1] and added their modifications to Alexnet to see if I could reproduce the same performance improvements. The major differences between ZF Net and Alexnet are in the first convolutional layer: a filter size of 7 rather than 11, a stride of 2 rather than 4, expanding the size of the middle convolutional layers and the number of filters used from 192 in each to 256/384. I did not adjust the number of filters because the included implementation of Alexnet already included this modification. See Figure 1 at end for more detail.

Further, I added training functionality to run the validation evaluation at multiple timesteps throughout training (ie. evaluate every 2500 iterations). This allowed me to have some insight into whether the model was overfitting on the training set. I wasn't sure how long the neural network needed to train, and if there were too many iterations, the performance might peak and then begin to decline as overfitting became an issue.

1.1.1 Results

The ZF Net modification had a training accuracy of top 1 – 46% and top 5 – 76% with a softmax cross-entropy error of 1.989. The valuation accuracy was top 1 – 37% and top 5 – 66% with a softmax cross-entropy error of 2.49. In the competition it was top 1 – 36%.

1.2 VGG Implementation

The second model I implemented was a number variations on "VGG Net" (2014) [2]. I chose to implement this architecture because it was the best performing model for which the initial authors trained on a single GPU, as opposed to many of the later models which, while better, needed vastly more computing power outside the budget given for the assignment.

The benefits of the architecture of VGG is as follows. Having two layers of 3x3 filters has an effective receptive field of 5x5. This simulates a larger filter while keeping the number of parameters low. The VGG paper contains 6 different models of different sizes, from 11-19 layers. I implemented the top two original best performing ones, a 19 and a 16 layer.

1.2.1 Memory experimentation

VGG proved difficult to train using the AWS cloud instance throughout the project. I first implemented the 19-layer model but it would not train because the EC2 instance ran out of VRAM. I tried then both the 16-layer and 11-layer (smallest) models, but neither successfully fit in GPU RAM with default training parameters. The issue of memory is reasonable, as VGG does contain far more parameters than the simple Alexnet.

I found a source online [3] that mentioned the removal some of the fully-connected layers would significantly reduce the model's parameters, though after implementation, this didn't seem to noticeably fix the memory issues.

By experimentation, I found that the main cause of the memory issue was the initial convolutional layer. I discovered this by combining the first layer of VGG with the remainder of the initial Alexnet implementation. While a small change from Alexnet, the lower stride size (1) on the VGG layer compared to Alexnet caused the parameter size and memory to blow up quite a bit, which carried down to other layers.

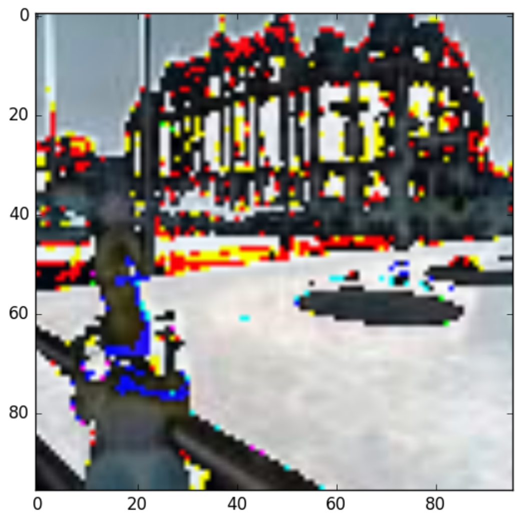
1.2.2 Training Speed

The only change that finally allowed the model to train was simply decreasing the batch size (cutting the memory usage significantly). This modified VGG Net was able to run, but at a rather slow speed: 1.7 iterations/second at 128 batch size. In terms of images, this is nearly 6x slower than Alexnet's training speed, and based on a variety of sources, VGG seems to require around 360,000 iterations, compared to Alexnet's 50,000. Without better hardware it wasn't feasible to finish the training within the deadline. Had it finished, it would have likely outperformed my first ZF Net architecture since it has historically proved to be a stronger model. The following analysis is on the first ZF Net implementation.

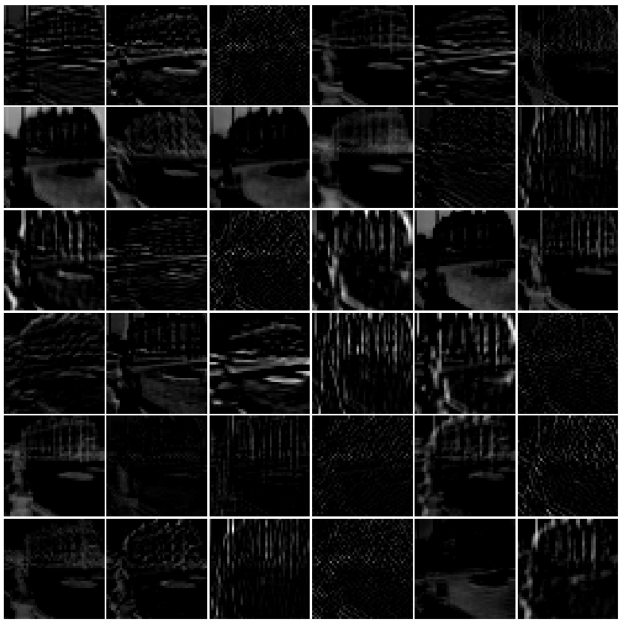
2 Visualization

I performed a visualization of the images, activations, and filters to get a better understanding of filters generated by the network. The following analysis steps through a single image traveling through the trained ZF Net model.

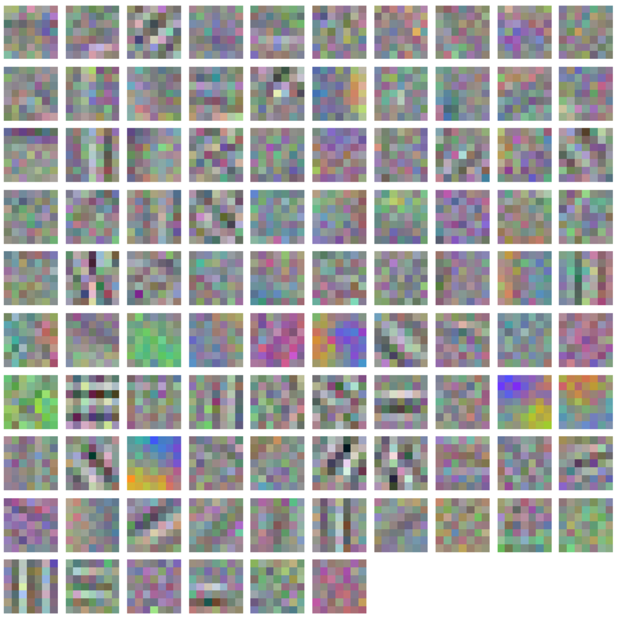
The image is first preprocessed by subtracting the mean of the data set, centering the data set at zero. This normalized image doesn't display well, but is better for the training process. It is likely that by normalizing the data, the system has an easier time optimizing for the data set.



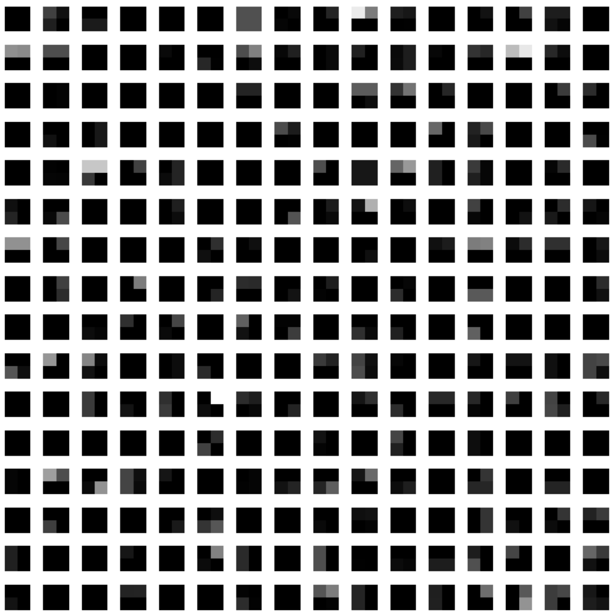
Next, I looked at the first layer output (convolutional layer 3x3 kernel, stride 2). The original image is still somewhat visible despite the convolutions. The responses here show many similarities to the manual oriented filters used in part 2 of this assignment. This image particularly seems to respond to horizontally oriented filters; note the strong white vertical lines on the house that have high horizontal gradient shifts.



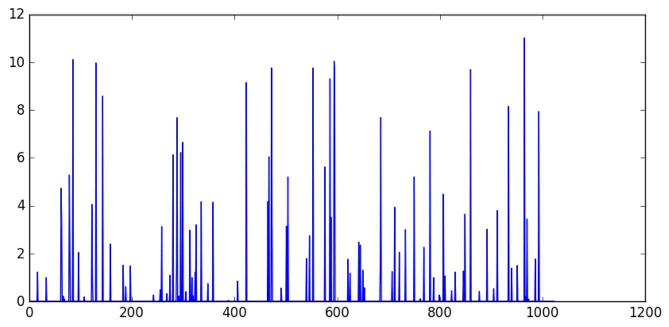
Next, I took a look at the filters of the first layer convolution. The presence of clear gradient filters suggests correct training. Some look very much like the stretched derivative of gaussian filters I used in part two of this homework, others look more like noise, which may indicate poor or insufficient training, or maybe a smaller model may have been sufficient.

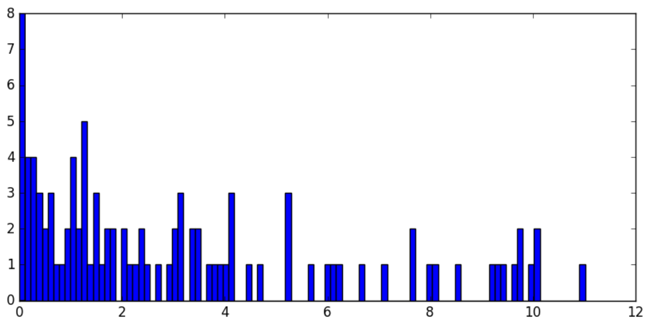


After the filters had been trained further, this is the output of the fifth layer after pooling. The activation grids are 2x2, and as expected, only a few of them activate strongly. This makes sense as further into the neural network, neurons become more specialized [4].



This shows the activation value outputs for the first fully connected layer, as well as a histogram of the activation values. Interestingly, the majority of the neurons seem to be activating at low values close to zero.





The following is the final output probabilities across all classes. The highest probability is .268 for class 0 which is "abbey", which is classified correctly. Other candidates include monastery/outdoor, church/outdoor, palace, and courtyard, which are all quite similar.

The following were the top 5 responses:

index	probability	label
0	0.2679885	abbey
67	0.184772	monastery/outdoor
32	0.17860141	church/outdoor
71	0.17546929	palace
42	0.077413633	courtyard

Figure 1: Overall Architecture Diagram

```
data      (256, 3, 96, 96)
label     (256,)
label_data_1_split_0      (256,)
label_data_1_split_1      (256,)
label_data_1_split_2      (256,)
conv1     (256, 96, 45, 45)
pool1     (256, 96, 22, 22)
conv2     (256, 256, 11, 11)
pool2     (256, 256, 5, 5)
conv3     (256, 384, 5, 5)
conv4     (256, 384, 5, 5)
conv5     (256, 256, 5, 5)
pool5     (256, 256, 2, 2)
fc6       (256, 1024)
fc7       (256, 1024)
fc8       (256, 100)
fc8_fc8_0_split_0         (256, 100)
fc8_fc8_0_split_1         (256, 100)
fc8_fc8_0_split_2         (256, 100)


Parameters:
conv1     (96, 3, 7, 7) (96,)
conv2     (256, 48, 5, 5) (256,)
conv3     (384, 256, 3, 3) (384,)
conv4     (384, 192, 3, 3) (384,)
conv5     (256, 192, 3, 3) (256,)
fc6       (1024, 1024) (1024,)
fc7       (1024, 1024) (1024,)
fc8       (100, 1024) (100,)
```

References

[1] Matthew D. Zeiler, Rob Fergus *Visualizing and Understanding Convolutional Networks* 2013.

[2] Karen Simonyan, Andrew Zisserman *Very Deep Convolutional Networks For Large-Scale Image Recognition* 2014: ICLR 2015

[3] Andrej Karpathy *CS 231n Convolutional Neural Networks for Image Recognition* 1940: http cs231n.github.ioconvolutional-networks#case

[4] Wikipedia contributors *Alternative Explanations of the "Grandmother" Cell* 2017: https://en.wikipedia.org/w/index.php?title=Alternative_explanations_of_the_%22grandmother%22_cell&oldid=769060098