**CS 291K Machine Problem 2 Report**

**Problem Statement:**

Implement a classifier as a convolutional neural network for use on the CIFAR-100 dataset. The classifier should identify 20 superclasses and output a 20 dimensional vector representing the predicted probabilities of an image belonging to each superclass.

**Implementation:**

I used TensorFlow to implement my convolutional network. In my program I defined some major functions:

conv2d: takes in tensors 4D input x and filter W and returns their 2D convolution (a tensorflow function). This function flattens the filter to a 2D matrix and extracts image patches from the input tensor to form a virtual tensor. For each patch, the function right-multiplies the filter matrix and the image patch vector. In my function, I set the stride size to 1.

max\_pool\_nxn: takes in a 4D tensor input, ksize, strides, padding type, and data format specification and performs max pooling on the input. For my function I used a ksize of 2.

one\_hot: takes in a vector of classifiers and returns a matrix where each classifier is a unique vector of zeros and ones.

l2: takes a tensor input and returns half of the l2 norm of the tensor without the square root.

Other functions were used directly from the TensorFlow library.

**Architecture:**

My architecture is as follows:

1. First convolutional layer with 12 feature maps
2. Max pooling layer
3. Second convolutional layer with 20 feature maps
4. Max pooling layer
5. Fully connected layer
6. Softmax loss classifier

**Model Building:**

I initialized the first layer weights with a normal distribution and a standard deviation of 0.001. The first layer biases were initialized to zero. Using greater than 20 feature maps did not significantly change my resulting accuracy, but did increase the training time. Using fewer than 20 feature maps significantly reduced training and test accuracy.

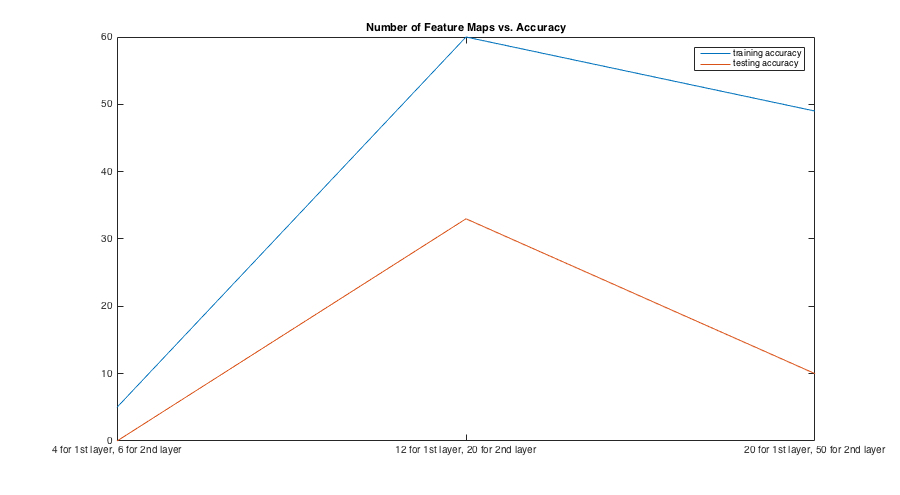
The second layer weights were initialized also with a normal distribution and standard deviation of 0.001. The second layer biases were also initialized to zero. I used 50 feature maps for my second layer, for the same reasons as above.

For training, I randomly select a batch of size 100 and run the convolutional network on the selected batch for each iteration.

**Results:**

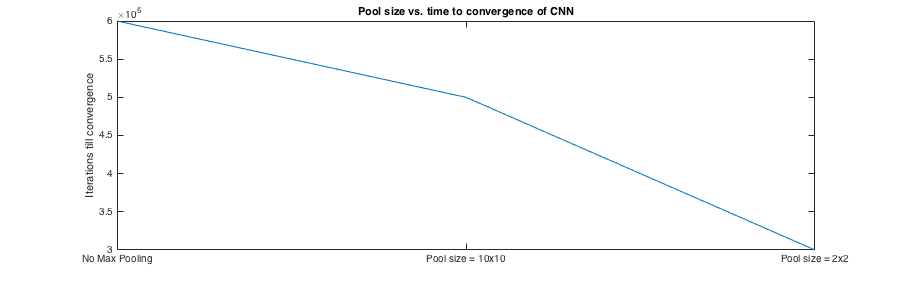
I was able to get 60% accuracy for the training sets and 33% accuracy for the testing set. Some empirical data follows.

|  |  |
| --- | --- |
| 1 Convolutional Layer | 2 Convolutional Layers |
| 10% Training Accuracy | 60% Training Accuracy |
| 0% Test Accuracy | 33% Test Accuracy |



**Extra Credits:**

1. I have implemented dropout, which decreased the training and test accuracy from 60% to 20% for training and 33% to 10% for testing. This is not a good option for my network.
2. Another optimization method I tried is the Adam Optimizer (tf.train.AdamOptimizer).
3. If all weights and biases are initialized to a small constant instead of randomly, the network will have much lower accuracy, down to 5% for training and 0% for testing accuracy.
4. Using smaller filter sizes resulted in greater accuracy, as shown in the Results section. However, training time took longer.
5. Using different filter sizes of the pooling layer didn’t really change the accuracy, but it did require more iterations for convergence. If I completely get rid of the max pooling step, the network runs extremely slowly.



**Challenges:**

It was difficult to pinpoint the right number of feature maps to use for each layer. It was also difficult to tune the initialization parameters for the weights and biases. Without enough feature maps, the training accuracy would stay around 5%, which is extremely low. However with too many feature maps, the network becomes very slow without improvement in accuracy.

**Possible Improvements:**

I could write a script to automatically fine tune the hyperparameters, which would be a lot more effective and efficient and finding the best hyperparameters to improve my accuracy. I could also train for a lot longer, though the improvement to accuracy would have diminishing returns. It could also help to add some extra layers to the convolutional neural network.