CIFAR-100 Using Convolutional Neural Nets

CS291K : Advanced Data Mining, MP1

Neeraj Kumar (9408485)

Introduction

The goal of this project is to implement an effective model for *classification* problems in image processing. We build a convolutional neural network to classify the images from CIFAR-100 dataset [1] into the 20 superclasses. The next few sections go into details of our implementation and discuss the observed results.

Implementation Details

Our neural network comprises of following layers:

- **Input Layer** L_1 The CIFAR-100 images are of the form $32 \times 32 \times 3$, they form the input layer with three channels.
- **Conv and MaxPool** L_2 The number of features at this level is the hyperparameter C_1. The output is $16 \times 16 \times C_1$. The image size is reduced by half due to maxpooling with 2×2 filter.
- **Conv and MaxPool** L_3 The number of features at this level is the hyperparameter C_2. As above, the output is $8 \times 8 \times C_2$.
- **Fully Connected** L_4 We know flatten the input and feed to a fully connected layer. The number of neurons is a hyperparameter N_{FC}.
- **Softmax Classifier** L_5 We use softmax for computing the loss and classifying the images into target classes.

The code structure is quite simple and follows the following architecture:

- The top level file is called `driver.py`. It basically preprocesses data, initializes the network and calls it on training and test sets.
- The file `convNet.py` does most of the stuff. It creates the computation graph G for tensorflow and then kickstarts the learning process by initializing the hyperparameters.

Model Building

In this section, we will discuss the hyperparameters for our model and how exactly we arrived at those values. The final values we settle for are shown in the following table. Note that most values we pick are powers of two, since they are easier to deal with.

Parameter	Value	Parameter	Value
Batch size	1000	Iterations	1000
Learning Rate	0.1	Decay	0.999
Conv1 Layers	8	Conv2 Layers	16
FC Layers	512	Regularization	1e-5

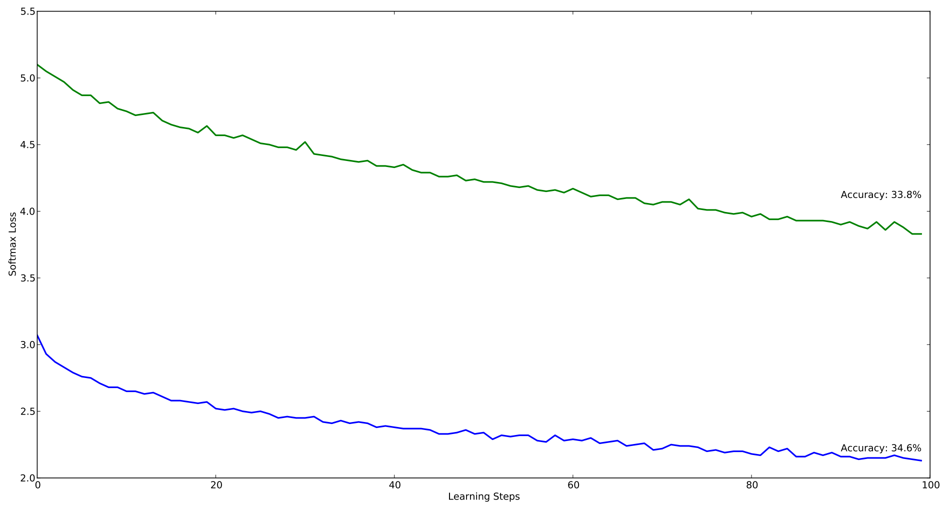

Figure 1: Plot of softmax loss over learning step. The green plot is with regularization, the blue one is without regularization.

These above values are based on the tradeoff bewteen accuracy and training time. One may increase the number of iterations and achieve better accuracies at the cost of higher training time. In the following we justify some of these choices.

1. **Number of feature maps** We found that more feature maps at each convolution level give better results, but at the cost of training time. See the results section.
2. **L2 Regularization** Since there are a lot of weight parameters in play, the regularization we should chose a small regularization. For the same setup as our best result, a regularization of `1e-3` gives us a training and testing accuracies around 36% and 32% respectively. Clearly, this is no better than our result without a regularization, so we chose a smaller value 1e-5.(See Figure 1).

Results

The following table gives a summary of our results. For all these results, the filter sizes for convolution and pooling were set to 5×5 and 2×2 respectively. The number of neurons in output layer are set to 512. The initial learning rate was picked to be 0.1. Our best result of 39.5% accuracy was obtained over 1000 iterations, without regularization.

Conv1	Conv2	Iterations	Training Acc.	Validation Acc.	Testing Acc.	Training Time
2	4	1000	30.9	29.7	29.4	22m
4	8	1000	34.2	32.44	30.6	28m18s
8	16	1000	43.3	39.5	38.0	32m18s

Extra Credits

In order to make the model more effective, we also added a few extra features. Note that in order to make data collection less time consuming, we only test some of these features for 100 iterations. However, it is normally the case that the behaviour depicted in early iterations is similar to the behavior later, so these are still good indicators.

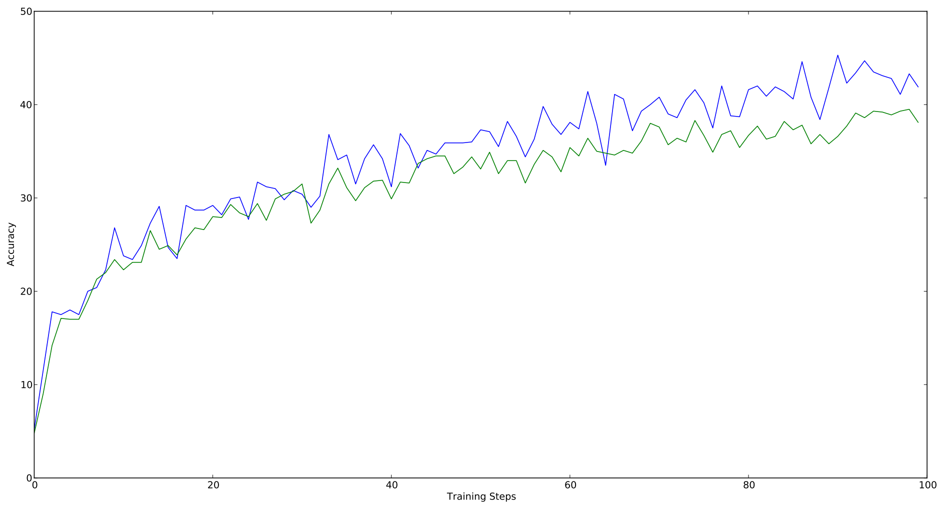

Figure 2: Accuracy plotted over 1000 iterations for our best set of hyperparameters. The blue and green lines represent the training and validation accuracies respectively.

- *Dropout* is a technique to reduce overfitting by randomly sampling neurons based on a threshold probability p and disabling them. The effects are shown in following table. Note that we apply dropout only for the training phase.

Probability	Training	Validation	Test
1	43.3	39.5	38.0
0.7	37.4	35.2	35.0
0.5	36.9	34.9	34.8

  It appears that dropout is not very effective in our case.

- *Convolution Filter* The default convolution filter is 5×5, we tried a few other values and the results are shown below.

Conv Filter	Training	Validation	Test	Time Taken
7×5	21.9	20.5	20.0	5m21s
5×5	23.8	22.3	22.1	3m11s
3×3	23.7	22.5	22.0	2m7s
1×4	13.4	13.2	11.2	48s

- *Neurons in FC layer* It is normally the case that more neurons you add to a FC layer, the better is the learning. But this usually comes at the cost of training time. In our case, the effects are not that pronounced because the convolution layers dominate the training time.

FC Size	Training	Validation	Test	Time Taken
32	14.5	14.9	14.0	1m41s
64	18.5	15.9	12.0	1m51s
128	20.3	17.9	18.0	1m38s
256	21.8	19.1	20.0	1m58s
512	23.7	22.5	22.0	2m17s

- *Optimization Technique* We try the following optimizers:

Optimizers	Initial Rate	Training	Validation	Test
Gradient Decent	0.1	23.7	22.5	22.0
Adam	1e-4	20.2	20.00	19.8
Momentum ($\mu = 0.5$)	0.1	22.8	21.3	20.0

- *Initialization* Tensorflow has quite a few initialization technique. We try a few of them as follows.

Initialization	Training	Validation	Test
Truncated Normal	23.7	22.5	22.0
Random Normal	22.1	20.6	19
Zeros	5.6	4.2	5

Observe that truncated normal (only pick values in 2 std. dev range) is the clear winner here.

Challenges and Improvements

One of the key challenges was finding the right number of feature maps at each layer. There were so many choices, and there is always a tradeoff in terms of training time and the accuracies. Also, preprocessing the image by normalizing was vital.

Likely improvements are better pre-processing techniques, PCA whitening, randomly cropping a region to get a fixed size smaller iamge, rotating the image etc. All these might give better accuracies.

References

[1] CIFAR-10 dataset. `https://www.cs.toronto.edu/~kriz/cifar.html`.