CSE253 Assignment3

Chihhung Lin
A53093906

Yi-Shu Tai
A53071463
yitai@eng.ucsd.edu

1 AWS

2 Load the data

3 Build your network

Our train_val.prototxt for the basic network is defined as follow:

```
net: "origin_network.prototxt"
test_iter: 100
test_interval: 500
base_lr: 0.001
momentum: 0.9
weight_decay: 0.004
lr_policy: "fixed"
display: 100
max_iter: 30000
snapshot: 4000
snapshot_prefix: "examples/cifar10/cifar10_quick"
solver_mode: GPU
```

Our origin_network.prototxt has the following structure:

Layer	Type	Input Size	Kernel Size	# Filters	Nonlinearity	Pooling	Stride	Size	Output Size	Parameters
1	Conv	32*32*3	5*5	32	ReLU	MAX	2	3*3	16*16*32	2,432
2	Conv	16*16*32	5*5	32	ReLU	AVE	2	3*3	8*8*32	25,632
3	Conv	8*8*32	5*5	64	ReLU	AVE	2	3*3	4*4*64	51,264
4	FC	4*4*64	1*1		ReLU				64*1	65,600
5	FC	64*1	1*1		Softmax				100*1	6,500

4 Train your network

Our training procedure is the listed as follow:

1. Download and convert cifar-100 to lmdb format
2. Define solver
3. Define network structure
4. Run "train –solver=solver.prototxt"
5. Check out the result

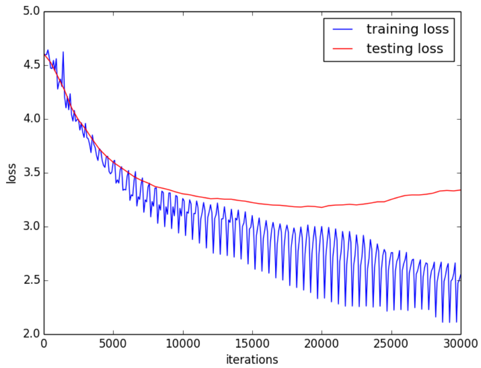

Figure 1: train-test loss vs iterations

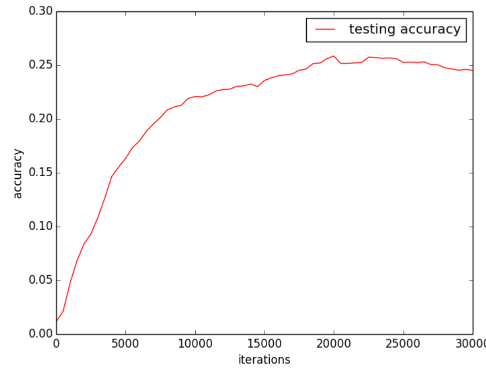

Figure 2: test accuracy vs iterations

5 Experiment with preprocessing the input data

(a) It took about 13000 iterations to hit the 0.39 accuracy, while the pervious one took about 20000 iterations to converage in accuracy. Also, its test accuracy outperformed the previous one (0.39 to 0.25).

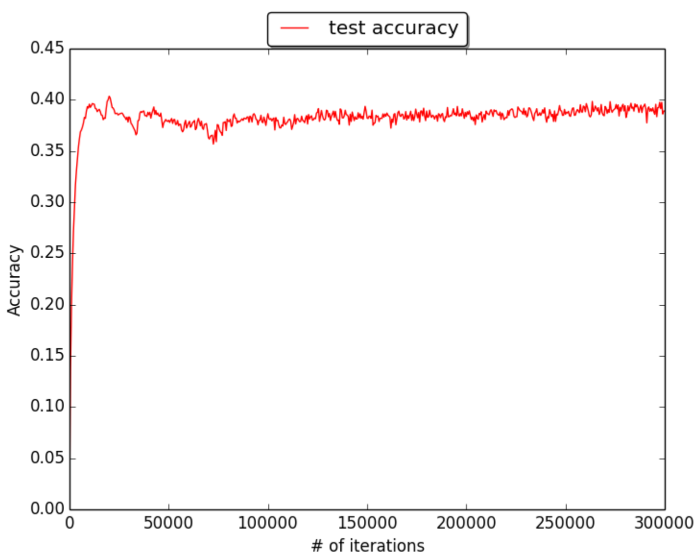

Figure 3: testing accuracy vs training iteration

(b) The performance went down as the number of images went down. We got 0.366 on 300 images per class and 0.26 on 100 images per class.

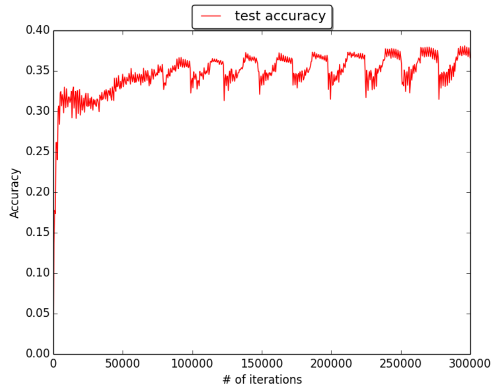

Figure 4: testing accuracy vs training iteration (300 images per class)

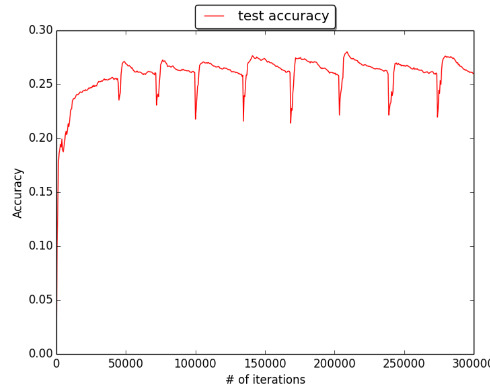

Figure 5: testing accuracy vs training iteration (100 images per class)

6 Experiment with optimization methods

(a) Stochastic Gradient Descent

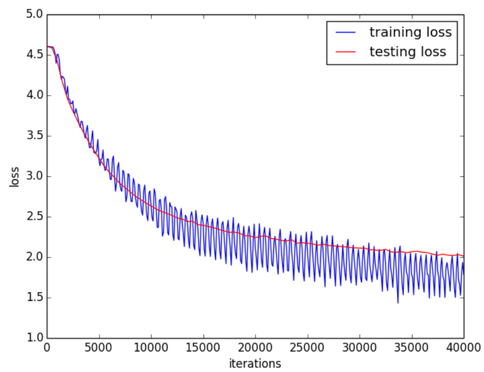

Figure 6: train-test loss vs iterations

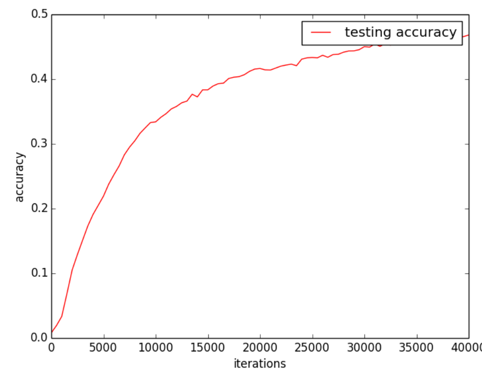

Figure 7: test accuracy vs iterations

(b) Adaptive Gradient Descent

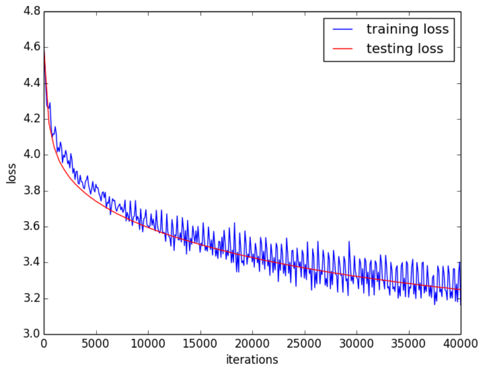

Figure 8: train-test loss vs iterations

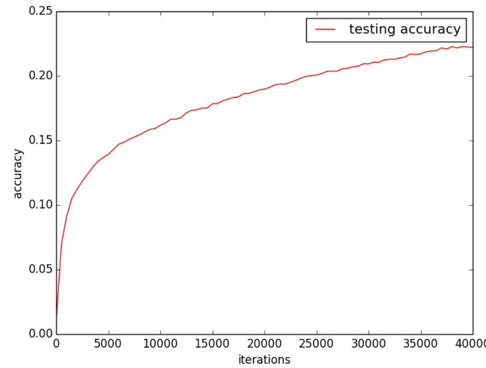

Figure 9: test accuracy vs iterations

(c) Nesterovs Accelerated Gradient

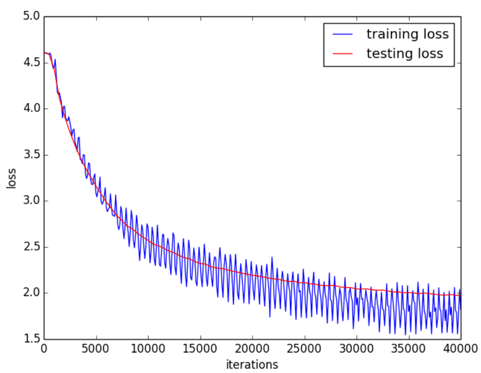

Figure 10: train-test loss vs iterations

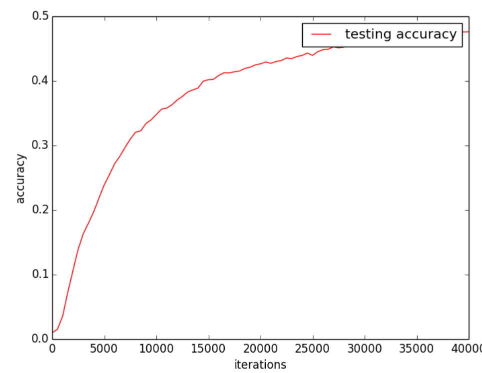

Figure 11: test accuracy vs iterations

(d) RMSprop

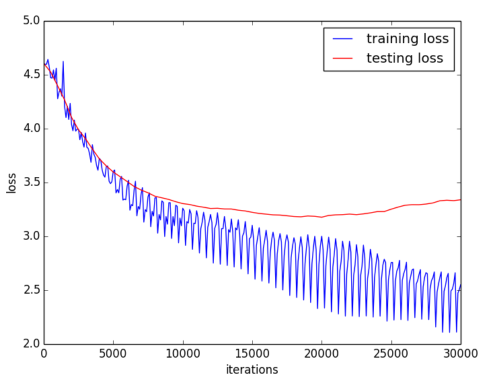

Figure 12: train-test loss vs iterations

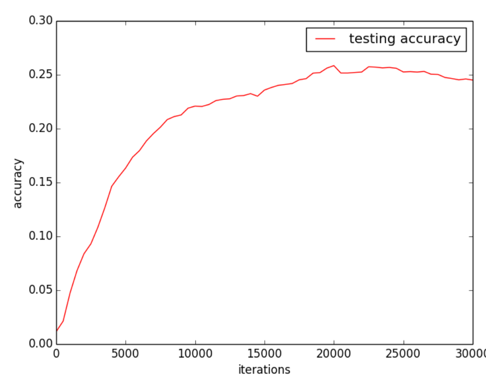

Figure 13: test accuracy vs iterations

7 Experiment with network structure

Our total parameters in origin network is 151,428. We create a new network as follow:

Layer	Type	Input Size	Kernel Size	# Filters	Nonlinearity	Pooling	Stride	Size	Output Size	Parameters
1	Conv	32*32*3	5*5	32	ReLU	MAX	2	3*3	16*16*32	2,432
2	Conv	16*16*32	5*5	32	ReLU	AVE	2	3*3	8*8*32	25,632
3	Conv	8*8*32	5*5	64	ReLU	AVE	2	3*3	4*4*64	51,264
4	FC	4*4*64	1*1		ReLU				32*1	32,800
5	FC	32*1	1*1		ReLU				128*1	4,224
6	FC	128*1	1*1		ReLU				128*1	16,512
7	FC	128*1	1*1		ReLU				64*1	8,256
8	FC	64*1	1*1		ReLU				64*1	4,160
9	FC	64*1	1*1		Softmax				100*1	6,500

The new network has 151,780 parameters, which is similar to our origin network.

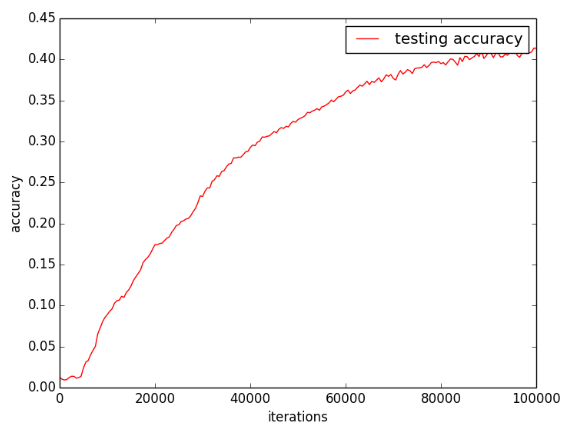

Figure 14: test accuracy vs iterations

As we can observe in the figure, with more hidden layers, the performance is roughly the same as the origin one. However, it takes more iterations to achieve similar accuracy comparing to the original network.

8 Experiment with network fine-tuning

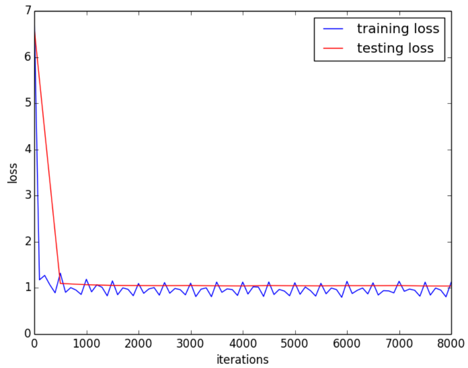

Figure 15: train-test loss vs iterations

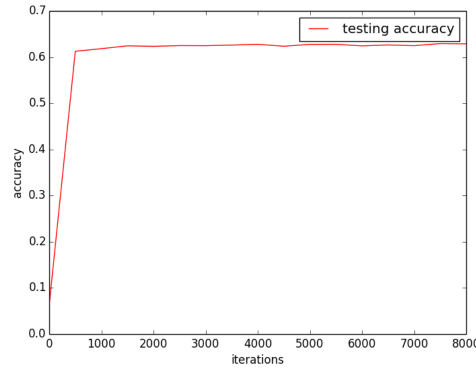

Figure 16: test accuracy vs iterations

As we can observe in the figure, with well-trained model, the network converges very fast and is generalize well.

9 Feature visualization

Figures below show the visualization of layer 1 (32 filters) and layer 2(32×32 filters) of our network. They look totally different.

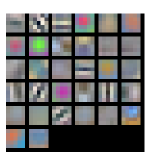

Figure 17: Feature visualization of layer 1

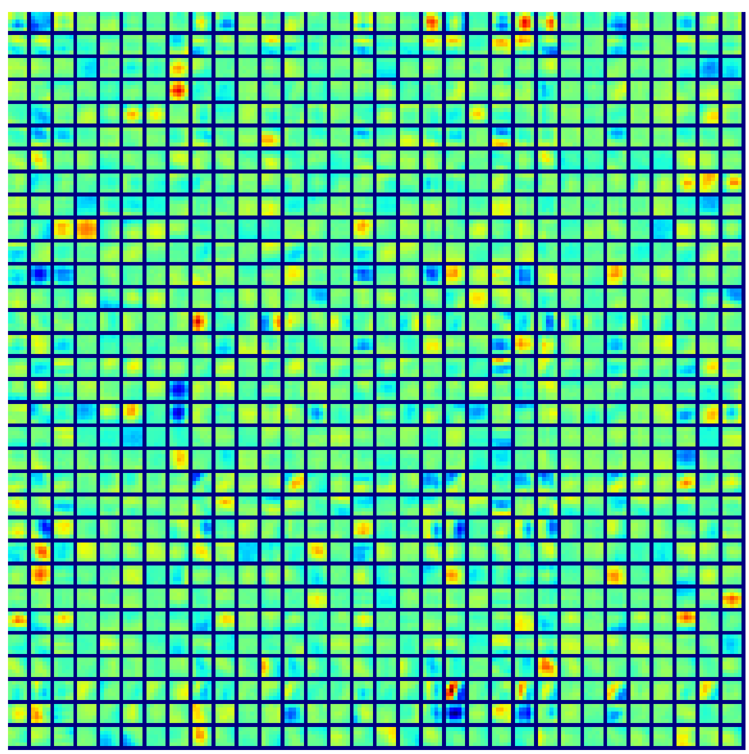

Figure 18: Feature visualization of layer 2