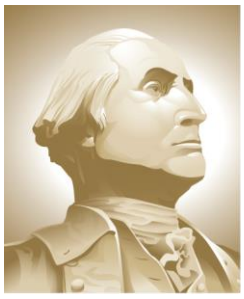


THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Machine Learning II
Exam #2
Prince Birring
Due: *Nov – 27- 2017*

Q1: Training Convolution networks with Caffe on MNSIT dataset

The objective of this miniproject is to become familiar with Caffe training deep convolution networks on large datasets. You will be provided with a sample program that loads in the MNSIT data set and sets up an example convolution network to be trained on it. (See http://caffe.berkeleyvision.org/gathered/examples/mnist.html for a discussion of this example.) The instructions below provide a rough framework of what you should do for the miniproject. You should experiment with using Caffe on this largest dataset. The project is open-ended. Learn as much as you can about be using deep convolution networks, and relate what you have learned in your project. For all parts below, and any other experiments you run, include the results into one PDF file, and upload it to the Blackboard. Include all program listings, plots, command line printouts, discussion etc.

1. **Download train_mnist.py, get_mnsit.py and create mnist.sh from** https://github.com/amir-jafari/Deep-Learning/tree/master/Caffe_/Mini_Project **and put them in the directory where you want to run your programs.**

```
princebirring1992@ubuntu-14:~$ git clone https://github.com/amir-jafari/Deep-Learning.git
Cloning into 'Deep-Learning'...
remote: Counting objects: 3195, done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 3195 (delta 0), reused 5 (delta 0), pack-reused 3187
Receiving objects: 100% (3195/3195), 100.81 MiB | 34.44 MiB/s, done.
Resolving deltas: 100% (222/222), done.
Checking connectivity... done.
```

```
princebirring1992@ubuntu-14:~$ ls
caffe                                                Deep-Learning-11-13
cuda                                                 Deep-Learning-11-23
cuda-repo-ubuntu1404-8-0-local-ga2_8.0.61-1_amd64.deb    demos
cuda-repo-ubuntu1404-8-0-local-ga2_8.0.61-1_amd64.deb.1  install-14-04-final1
cudnn-8.0-linux-x64-v6.0.tgz                         log
cudnn-8.0-linux-x64-v6.0.tgz.1                       NLP
data                                                 nltk_data
Deep-Learning                                        pycharm-community_20
princebirring1992@ubuntu-14:~$ cd Deep-Learning
princebirring1992@ubuntu-14:~/Deep-Learning$ ls
Caffe_  Keras_  Readme.md  Tenflow_  Theano_  Torch_
princebirring1992@ubuntu-14:~/Deep-Learning$ cd Caffe_/
princebirring1992@ubuntu-14:~/Deep-Learning/Caffe_$ ls
Caffe_Documentation  Create_LMDB  Mini_Project  Readme.md  Simple_Example  Sq
princebirring1992@ubuntu-14:~/Deep-Learning/Caffe_$ cd Mini_Project/
princebirring1992@ubuntu-14:~/Deep-Learning/Caffe_/Mini_Project$ ls
create_mnist.sh  get_mnist.sh  lenet_solver.prototxt  lenet_train_test.prototx
princebirring1992@ubuntu-14:~/Deep-Learning/Caffe_/Mini_Project$
```

2. **Run the following commands on terminal in your working path directory:**

   chmod 777 create_mnist.sh

   chmod 777 get_mnist.sh

```
princebirring1992@ubuntu-14:~/Deep-Learning/Caffe_/Mini_Project$ ls
create_mnist.sh  get_mnist.sh  lenet_solver.prototxt  lenet_train_test.prototxt  train_mnist.py
princebirring1992@ubuntu-14:~/Deep-Learning/Caffe_/Mini_Project$ chmod 777 create_mnist.sh
princebirring1992@ubuntu-14:~/Deep-Learning/Caffe_/Mini_Project$ chmod 777 get_mnist.sh
princebirring1992@ubuntu-14:~/Deep-Learning/Caffe_/Mini_Project$
```

./get_mnist.sh

```
princebirring1992@ubuntu-14:~/Deep-Learning/Caffe_/Mini_Project$ ./get_mnist.sh
Downloading...
--2017-11-24 02:28:32--  http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Resolving yann.lecun.com (yann.lecun.com)... 216.165.22.6
Connecting to yann.lecun.com (yann.lecun.com)|216.165.22.6|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 9912422 (9.5M) [application/x-gzip]
Saving to: 'train-images-idx3-ubyte.gz'

100%[=======================================================================================

2017-11-24 02:28:33 (32.3 MB/s) - 'train-images-idx3-ubyte.gz' saved [9912422/9912422]

--2017-11-24 02:28:33--  http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Resolving yann.lecun.com (yann.lecun.com)... 216.165.22.6
Connecting to yann.lecun.com (yann.lecun.com)|216.165.22.6|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 28881 (28K) [application/x-gzip]
Saving to: 'train-labels-idx1-ubyte.gz'
```

3. **Open the create_mnist.sh and change the path to EXAMPLE and DATA to the directory that you are working with. Then run the following commands.**

Change the Directory of Example, data and build in create_mnist.sh

```
EXAMPLE=/home/ajafari/Deep-Learning/Caffe_/Mini_Project
DATA=/home/ajafari/Deep-Learning/Caffe_/Mini_Project
BUILD=/home/ajafari/caffe/build/examples/mnist
```

TO:

```
EXAMPLE=/home/princebirring1992/Deep-Learning/Caffe_/Mini_Project
DATA=/home/princebirring1992/Deep-Learning/Caffe_/Mini_Project
BUILD=/home/princebirring1992/caffe/build/examples/mnist
```

./create_mnist.sh

```
princebirring1992@ubuntu-14:~/Deep-Learning/Caffe_/Mini_Project$ ./create_mnist.sh
Creating lmdb...
libdc1394 error: Failed to initialize libdc1394
I1124 02:33:30.425961  2087 db_lmdb.cpp:35] Opened lmdb /home/princebirring1992/Deep
I1124 02:33:30.426306  2087 convert_mnist_data.cpp:88] A total of 60000 items.
I1124 02:33:30.426313  2087 convert_mnist_data.cpp:89] Rows: 28 Cols: 28
I1124 02:33:30.967754  2087 convert_mnist_data.cpp:108] Processed 60000 files.
libdc1394 error: Failed to initialize libdc1394
I1124 02:33:33.996101  2102 db_lmdb.cpp:35] Opened lmdb /home/princebirring1992/Deep
I1124 02:33:33.996421  2102 convert_mnist_data.cpp:88] A total of 10000 items.
I1124 02:33:33.996430  2102 convert_mnist_data.cpp:89] Rows: 28 Cols: 28
I1124 02:33:34.083007  2102 convert_mnist_data.cpp:108] Processed 10000 files.
Done.
```

4. **By this time, you should have the train and test lmdb files of the mnist data set.**

```
/Caffe_/Mini_Project$ ls
 mnist_test_lmdb    t10k-images-idx3-ubyte  train-images-idx3-ubyte
 mnist_train_lmdb   t10k-labels-idx1-ubyte  train-labels-idx1-ubyte
/Caffe_/Mini_Project$
```

5. **Download lenet_solver.protxt, lenet_train.protxt from GitHub and put them in the directory. Change the path of net in lenet_solver.protxt to your working path directory.**

```
princebirring1992@ubuntu-14:~/Deep-Learning
create_mnist.sh   lenet_solver.prototxt
get_mnist.sh      lenet_train_test.prototxt
```

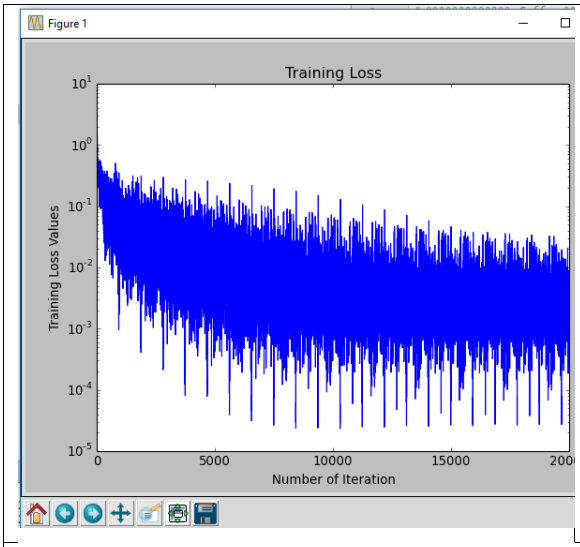
Change the path in lenet_sover.protext

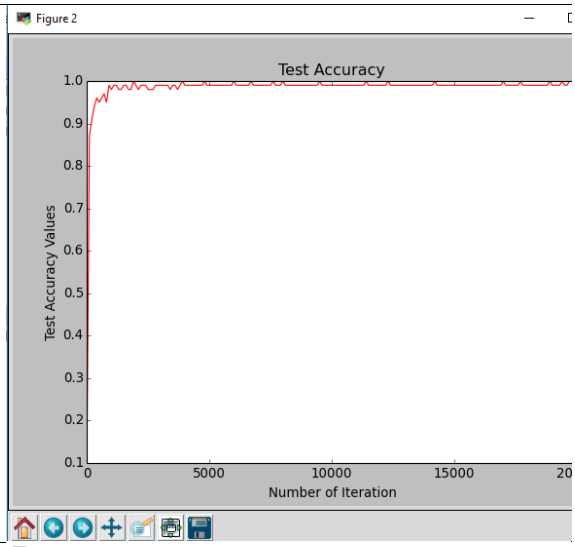
```
# The train/test net protocol buffer definition
net: "lenet_train_test.prototxt"
# test_iter specifies how many forward passes the test
# In the case of MNIST, we have test batch size 100 and
# covering the full 10,000 testing images.
```

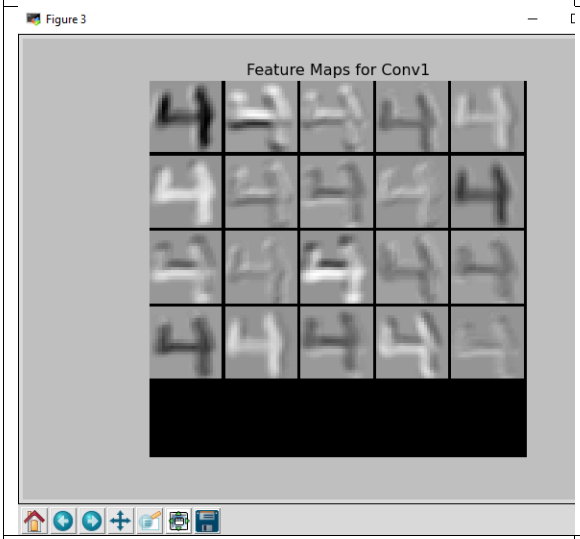
To:

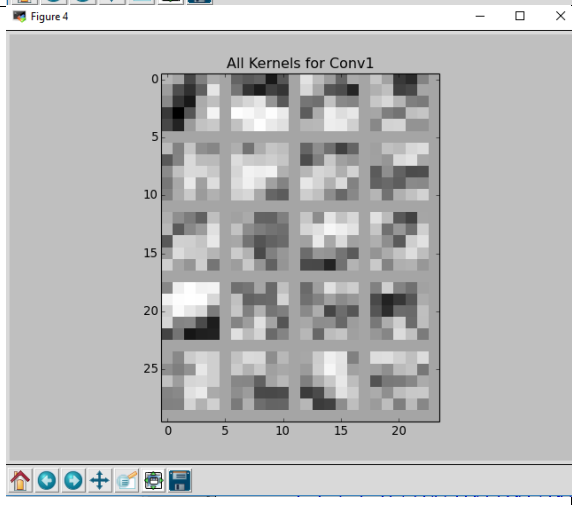
```
# The train/test net protocol buffer definition
net: "/home/princebirring1992/Deep-Learning/Caffe_/Mini_Project/lenet_train_test.prototxt"
# test_iter specifies how many forward passes the test should carry out.
# In the case of MNIST, we have test batch size 100 and 100 test iterations,
# covering the full 10,000 testing images.
```

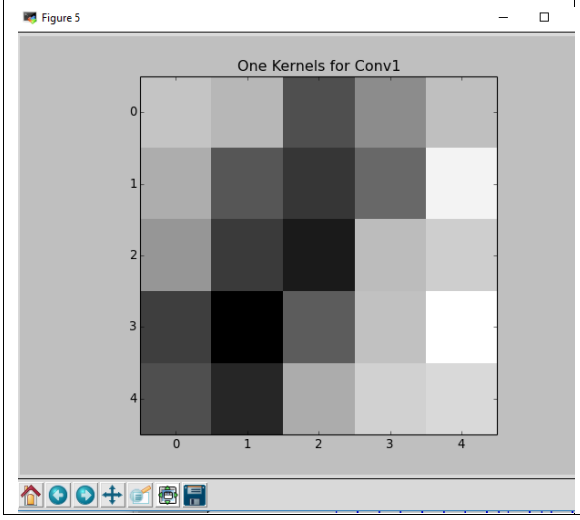
6. **Run the program train_mnist.py in PyCharm and investigate and verify its performance. You may need to change the line "my_root =" to the appropriate path.**












```
data    (64, 1, 28, 28)
label   (64,)
conv1   (64, 20, 24, 24)
pool1   (64, 20, 12, 12)
conv2   (64, 50, 8, 8)
pool2   (64, 50, 4, 4)
ip1 (64, 500)
ip2 (64, 10)
loss    ()
conv1   (20, 1, 5, 5) (20,)
conv2   (50, 20, 5, 5) (50,)
ip1 (500, 800) (500,)
ip2 (10, 500) (10,)

Process finished with exit code 0
```

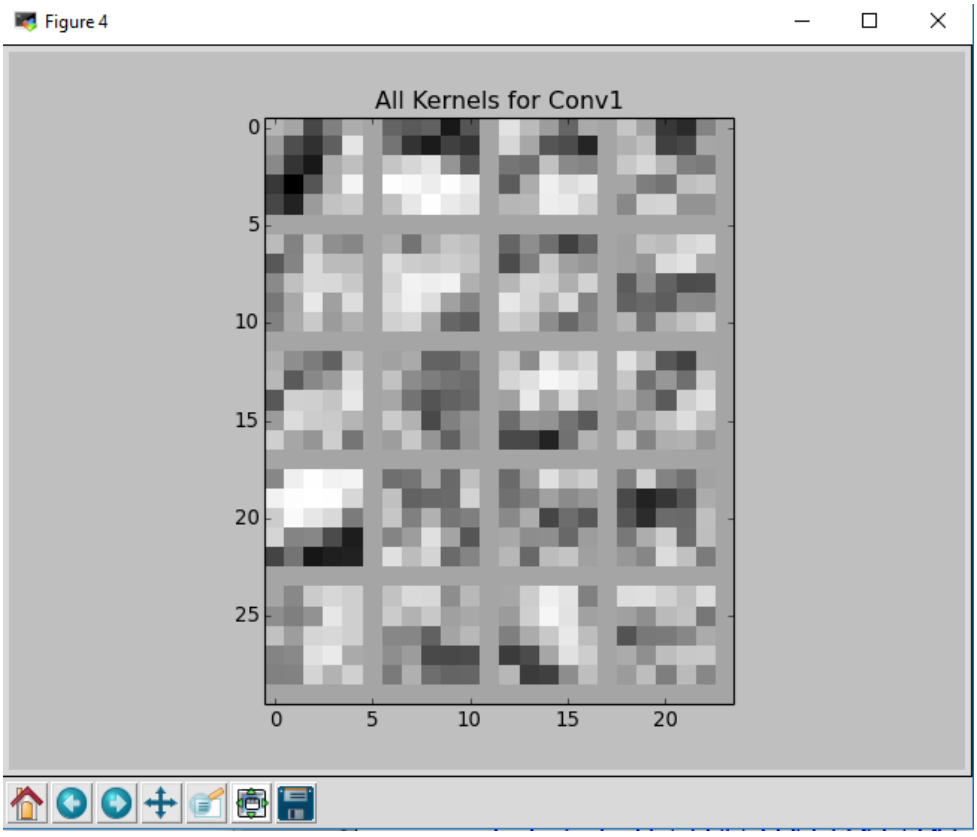
7. **Investigate the kernels in two convolution layers. Can you identify kernels that would be useful for particular numerals?**

Kernel is an integral component of the layered architecture. It is used to extract feature from input data. In two convolution layers, the first convolution layer has 20 kernels and the second one has 50 kernels.

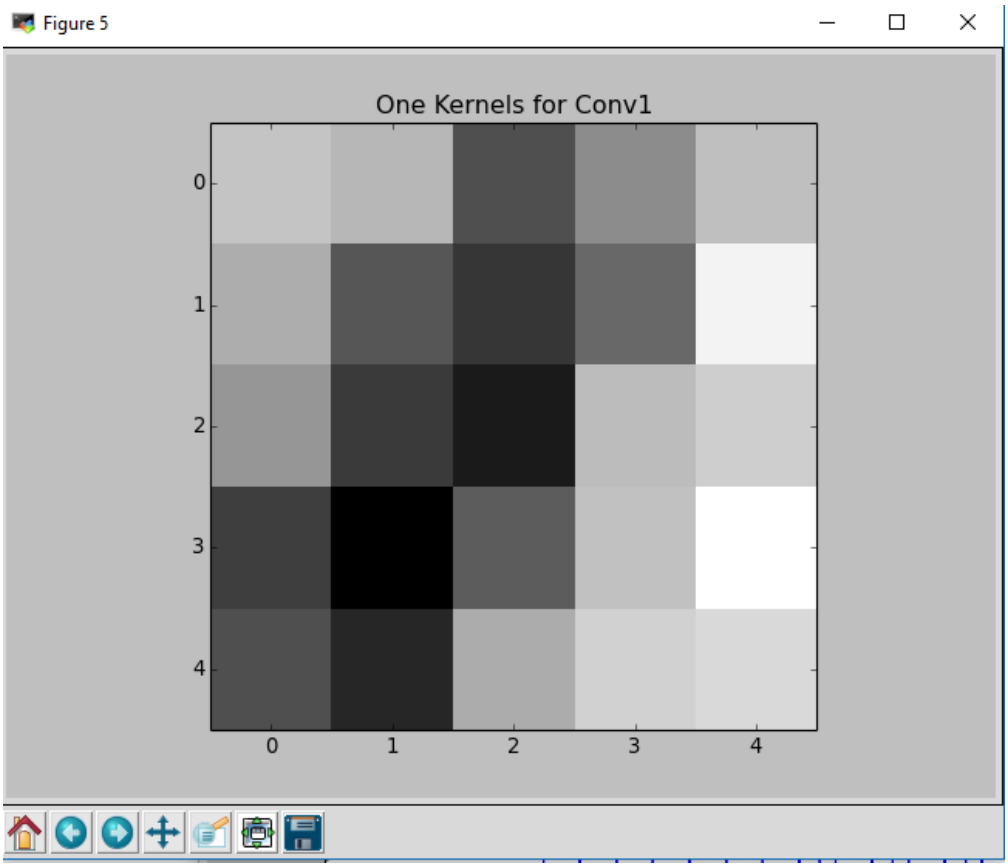
```
data    (64, 1, 28, 28)
label   (64,)
conv1   (64, 20, 24, 24)
pool1   (64, 20, 12, 12)
conv2   (64, 50, 8, 8)
pool2   (64, 50, 4, 4)
ip1 (64, 500)
ip2 (64, 10)
loss    ()
conv1   (20, 1, 5, 5) (20,)
conv2   (50, 20, 5, 5) (50,)
ip1 (500, 800) (500,)
ip2 (10, 500) (10,)

Process finished with exit code 0
```

First Layer with 20 Kernels (conv1): Different Kernels try to extract different feature from the input data. The below kernels have the edges such as 4.

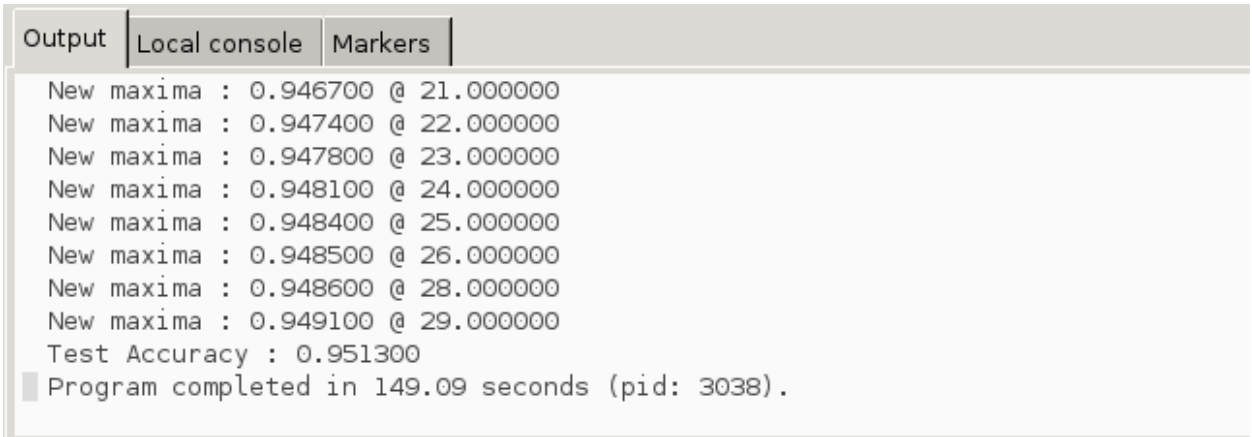


One Kernel from conv1:



8. How does the performance of the convolution network compare with the multilayer network works that you used in Exam#1?

MLP:



CNN:

```
Iteration 19900 testing... accuracy: 1.0
data    (64, 1, 28, 28)
label   (64,)
conv1   (64, 20, 24, 24)
pool1   (64, 20, 12, 12)
conv2   (64, 50, 8, 8)
pool2   (64, 50, 4, 4)
ip1 (64, 500)
ip2 (64, 10)
loss    ()
conv1   (20, 1, 5, 5) (20,)
conv2   (50, 20, 5, 5) (50,)
ip1 (500, 800) (500,)
ip2 (10, 500) (10,)
Time: 162.848049164 seconds
```

Multilayer Perceptron network trains the networks faster than convolution network. Whereas the test accuracy of the Convolution network is better than multilayer perceptron network.

9. **Change the size of the minibatches (batch_size paramter). If you make the batch size very large, does it affect the computation time significantly? Describe the advantages and disadvantages of increasing the batch size. Find a good choice.**

- **Batch Size = 8**

```
Iteration 19900 testing... accuracy: 1.0
data    (8, 1, 28, 28)
label   (8,)
conv1   (8, 20, 24, 24)
pool1   (8, 20, 12, 12)
conv2   (8, 50, 8, 8)
pool2   (8, 50, 4, 4)
ip1 (8, 500)
ip2 (8, 10)
loss    ()
conv1   (20, 1, 5, 5) (20,)
conv2   (50, 20, 5, 5) (50,)
ip1 (500, 800) (500,)
ip2 (10, 500) (10,)
Time: 124.810389042 seconds
```

- **Batch Size = 16**

```
Iteration 19900 testing... accuracy: 1.0
data    (16, 1, 28, 28)
label   (16,)
conv1   (16, 20, 24, 24)
pool1   (16, 20, 12, 12)
conv2   (16, 50, 8, 8)
pool2   (16, 50, 4, 4)
ip1 (16, 500)
ip2 (16, 10)
loss    ()
conv1   (20, 1, 5, 5) (20,)
conv2   (50, 20, 5, 5) (50,)
ip1 (500, 800) (500,)
ip2 (10, 500) (10,)
Time: 121.33317709 seconds
```

- **Batch Size = 32**

```
Iteration 19900 testing... accuracy: 0.980000019073
data    (32, 1, 28, 28)
label   (32,)
conv1   (32, 20, 24, 24)
pool1   (32, 20, 12, 12)
conv2   (32, 50, 8, 8)
pool2   (32, 50, 4, 4)
ip1 (32, 500)
ip2 (32, 10)
loss    ()
conv1   (20, 1, 5, 5) (20,)
conv2   (50, 20, 5, 5) (50,)
ip1 (500, 800) (500,)
ip2 (10, 500) (10,)
Time: 123.823718071 seconds
```

- **Batch Size = 64(default)**

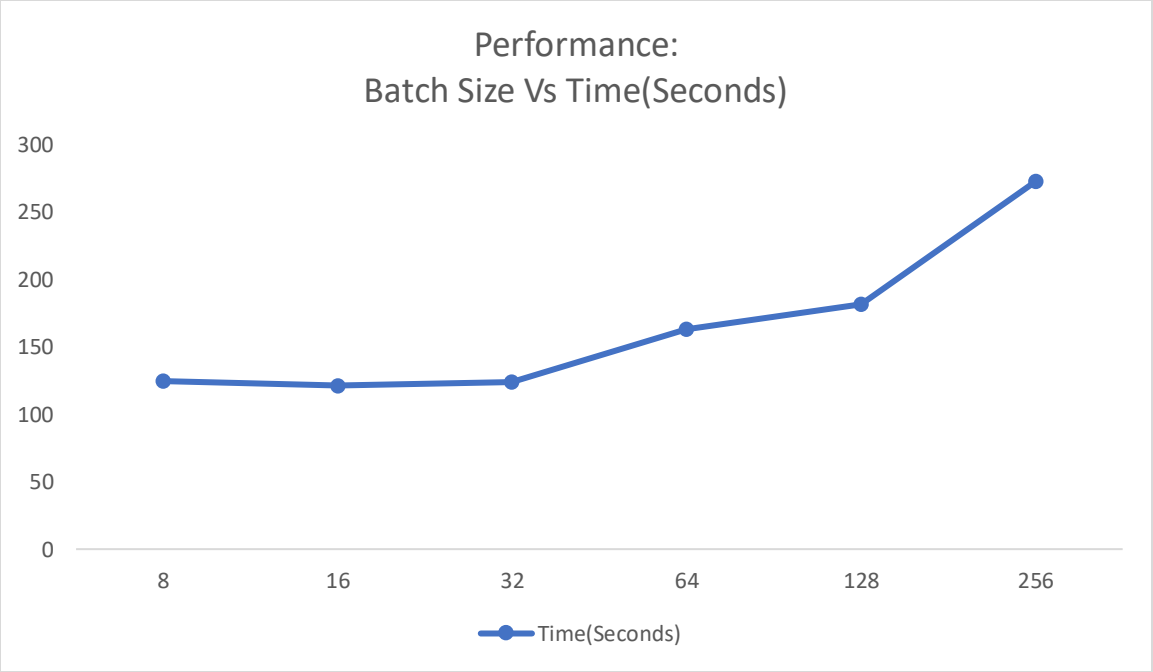
```
Iteration 19900 testing... accuracy: 1.0
data    (64, 1, 28, 28)
label   (64,)
conv1   (64, 20, 24, 24)
pool1   (64, 20, 12, 12)
conv2   (64, 50, 8, 8)
pool2   (64, 50, 4, 4)
ip1 (64, 500)
ip2 (64, 10)
loss    ()
conv1   (20, 1, 5, 5) (20,)
conv2   (50, 20, 5, 5) (50,)
ip1 (500, 800) (500,)
ip2 (10, 500) (10,)
Time: 162.848049164 seconds
```

- **Batch Size = 128**

```
Iteration 19900 testing... accuracy: 1.0
data    (128, 1, 28, 28)
label   (128,)
conv1   (128, 20, 24, 24)
pool1   (128, 20, 12, 12)
conv2   (128, 50, 8, 8)
pool2   (128, 50, 4, 4)
ip1 (128, 500)
ip2 (128, 10)
loss    ()
conv1   (20, 1, 5, 5) (20,)
conv2   (50, 20, 5, 5) (50,)
ip1 (500, 800) (500,)
ip2 (10, 500) (10,)
Time: 181.262885094 seconds
```

- **Batch Size = 256**

```
data    (256, 1, 28, 28)
label   (256,)
conv1   (256, 20, 24, 24)
pool1   (256, 20, 12, 12)
conv2   (256, 50, 8, 8)
pool2   (256, 50, 4, 4)
ip1 (256, 500)
ip2 (256, 10)
loss    ()
conv1   (20, 1, 5, 5) (20,)
conv2   (50, 20, 5, 5) (50,)
ip1 (500, 800) (500,)
ip2 (10, 500) (10,)
Time: 272.481292009 seconds
```

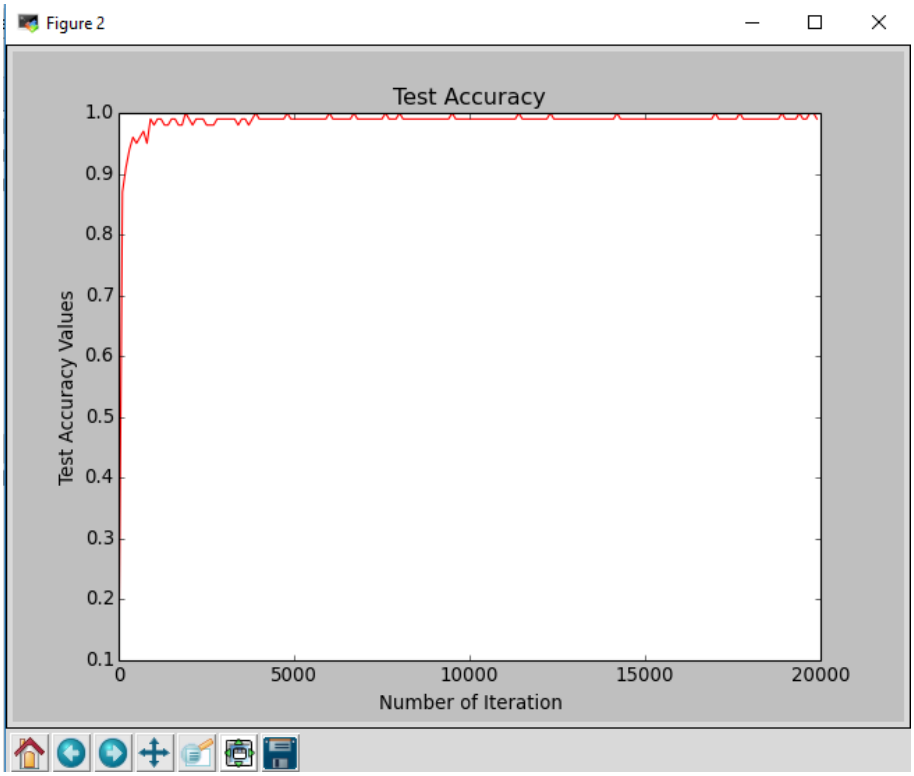


Advantages: As the batch size Increase the accuracy increases**.**

Disadvantage: As the batch size Increases the execution timing also increase (Slower training).

10. **Use the dropout layer fc1. Make fc1 the top and bottom for the dropout layer. (See** https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf **for a description of dropout) Does dropout improve the testing error?**

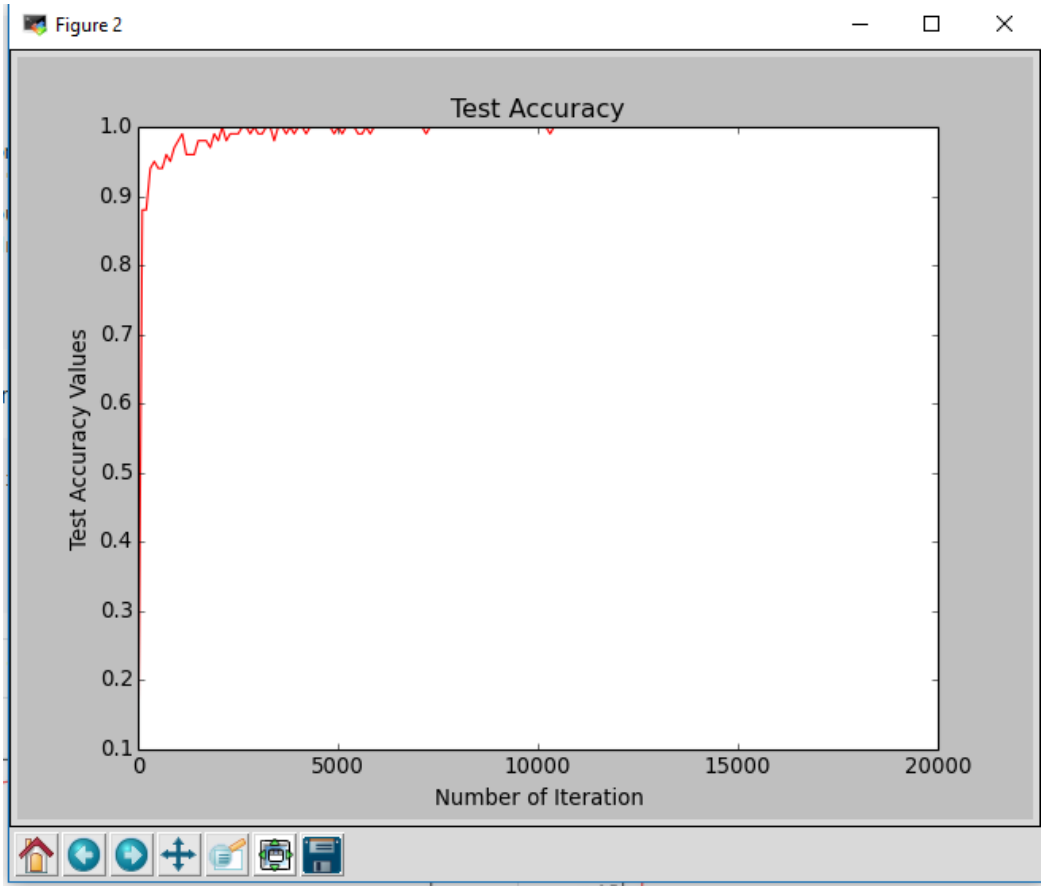
Non-Dropout Layer:




```
Iteration 19900 testing... accuracy: 1.0
data    (64, 1, 28, 28)
label   (64,)
conv1   (64, 20, 24, 24)
pool1   (64, 20, 12, 12)
conv2   (64, 50, 8, 8)
pool2   (64, 50, 4, 4)
ip1 (64, 500)
ip2 (64, 10)
loss    ()
conv1   (20, 1, 5, 5) (20,)
conv2   (50, 20, 5, 5) (50,)
ip1 (500, 800) (500,)
ip2 (10, 500) (10,)
Time: 162.848049164 seconds
```

Dropout Layer:

```
layer{
name: "fc1"
type:"Dropout"
bottom:"ip1"
top:"ip1"
    dropout_param{
        dropout_ratio: 0.6
    }
}
```




```
Iteration 19900 testing... accuracy: 1.0
data    (64, 1, 28, 28)
label   (64,)
conv1   (64, 20, 24, 24)
pool1   (64, 20, 12, 12)
conv2   (64, 50, 8, 8)
pool2   (64, 50, 4, 4)
ip1 (64, 500)
ip2 (64, 10)
loss    ()
conv1   (20, 1, 5, 5) (20,)
conv2   (50, 20, 5, 5) (50,)
ip1 (500, 800) (500,)
ip2 (10, 500) (10,)
Time: 192.588617086 seconds
```

In dropout layer the test accuracy increase but it takes more time to train the model as compare to without dropout layer.

11. **Experiment with different numbers of layers and different numbers of kernels. Maintain the total number of weights and biases in the network, while increasing the number of layers in network. Describe how the performance changes as the number of layers increases – both in terms of training time and performance.**

Default lenet network has two convolution layers and two pooling layers. I decreased the lenet network to one convolution layer and one pooling layer. Increased the number of kernels to 100.

Result: The Training Timing Increases from 163 seconds to 886 second and the test accuracy also decreases from 1.0 to 0.99.

- **Output**

```
Iteration 19900 testing... accuracy: 0.990000009537
data    (64, 1, 28, 28)
label   (64,)
conv1   (64, 100, 24, 24)
pool1   (64, 100, 12, 12)
ip2 (64, 10)
loss    ()
conv1   (100, 1, 5, 5) (100,)
ip2 (10, 14400) (10,)
Time: 886.147856951 seconds
```

- **Training Accuracy and Training Loss Graph**

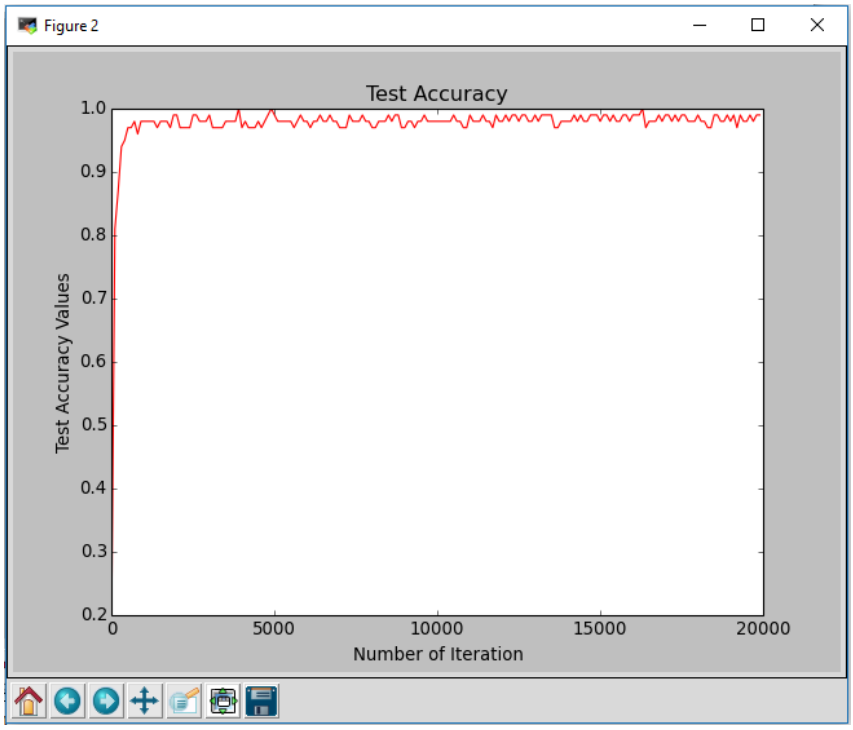


Figure 2
Test Accuracy
Test Accuracy Values
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
5000
10000
15000
20000
Number of Iteration

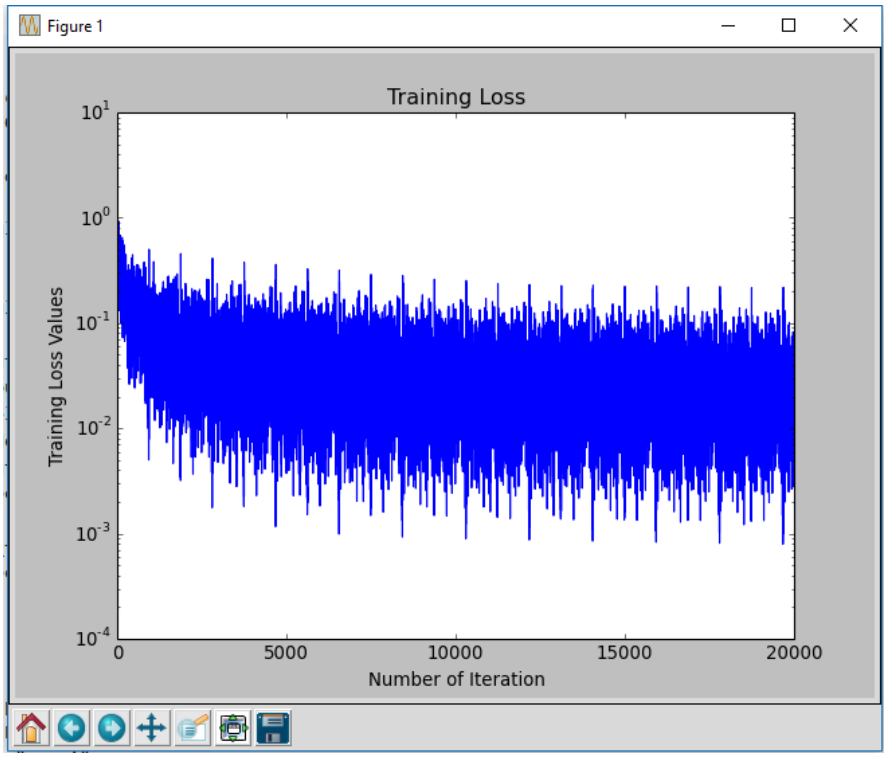


Figure 1
Training Loss
Training Loss Values
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
0
5000
10000
15000
20000
Number of Iteration

12. **Try one other training function from the list on this page:** http://caffe.berkeleyvision.org/tutorial/solver.html **Compare the performance with gradient descent.**

- **Lenet_solver.prototxt**

```
#snapshot_prefix: "examples/mnist/lenet_adadelta"
# solver mode: CPU or GPU
solver_mode: GPU
type: "AdaDelta"
delta: 1e-6
```

- **Train_mnist.py**

```
#solver = caffe.get_solver('lenet_solver.prototxt')
# Use SGDSolver, namely stochastic gradient descent algorithm
#solver = caffe.SGDSolver('lenet_solver.prototxt')
solver = caffe.AdaDeltaSolver('lenet_solver.prototxt')
#----------You need to run the following command to goustat works-
#  sudo pip install gpustat
```

- **Output**

```
Iteration 19900 testing... accuracy: 0.990000009537
data    (64, 1, 28, 28)
label   (64,)
conv1   (64, 50, 24, 24)
pool1   (64, 50, 12, 12)
conv2   (64, 20, 8, 8)
pool2   (64, 20, 4, 4)
ip1 (64, 500)
ip2 (64, 10)
loss    ()
conv1   (50, 1, 5, 5) (50,)
conv2   (20, 50, 5, 5) (20,)
ip1 (500, 320) (500,)
ip2 (10, 500) (10,)
Time: 190.605648041 seconds
```

- **Test Accuracy and Training Loss Graph**