

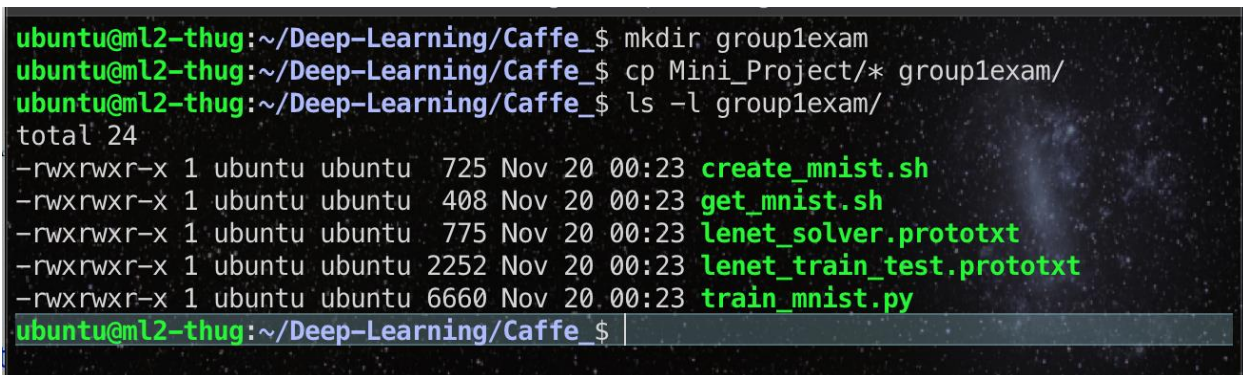
## EXAM 2

**Question 1: Download train\_mnist.py, get\_mnist.sh and create\_mnist.sh from GitHub and put them in your working directory.**

We already had the files in our cloud accounts. We obtained it when we cloned the deep learning repository from GitHub. Our group decided to make a new directory in the Caffe directory from our cloud. The directory path for our, newly created, exam directory looks something like this:

*/home/ubuntu/Deep-Learning/Caffe\_/group1exam/*

The files we needed were in the Mini\_Project folder. Using shell commands, we simply copied all the folders in the Mini\_Project directory to our exam folder. Figure 1 presents the input command in terminal and the listed files in the exam folder after the commands are entered.



```
ubuntu@ml2-thug:~/Deep-Learning/Caffe_$ mkdir group1exam
ubuntu@ml2-thug:~/Deep-Learning/Caffe_$ cp Mini_Project/* group1exam/
ubuntu@ml2-thug:~/Deep-Learning/Caffe_$ ls -l group1exam/
total 24
-rwxrwxr-x 1 ubuntu ubuntu 725 Nov 20 00:23 create_mnist.sh
-rwxrwxr-x 1 ubuntu ubuntu 408 Nov 20 00:23 get_mnist.sh
-rwxrwxr-x 1 ubuntu ubuntu 775 Nov 20 00:23 lenet_solver.prototxt
-rwxrwxr-x 1 ubuntu ubuntu 2252 Nov 20 00:23 lenet_train_test.prototxt
-rwxrwxr-x 1 ubuntu ubuntu 6660 Nov 20 00:23 train_mnist.py
ubuntu@ml2-thug:~/Deep-Learning/Caffe_$
```

*Figure 1: Getting the exam files using terminal*

**Question 2: Change the permission to 777 for the script files.**

Instead of individually changing the permission for the files, we approached the problem using by using the recursive flag (-R) on our exam folder. This will change the permission for the files in the exam directory at the same time. After changing the permissions for the folder, the get\_mnist.sh script was initiated. Figure 2 illustrates the process and the output of running the get\_mnist.sh script.

```

ubuntu@ml2-thug:~/Deep-Learning/Caffe_$ chmod -R 777 group1exam/
ubuntu@ml2-thug:~/Deep-Learning/Caffe_$ cd group1exam/
ubuntu@ml2-thug:~/Deep-Learning/Caffe_/group1exam$ ./get_mnist.sh
Downloading...
--2018-11-20 00:25:38-- 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'

train-images-idx3-u 100%[=====] 9.45M 37.1MB/s in 0.3s

2018-11-20 00:25:38 (37.1 MB/s) - 'train-images-idx3-ubyte.gz' saved [9912422/9912422]

--2018-11-20 00:25:38-- 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'

train-labels-idx1-u 100%[=====] 28.20K --KB/s in 0.03s

2018-11-20 00:25:38 (1.07 MB/s) - 'train-labels-idx1-ubyte.gz' saved [28881/28881]

--2018-11-20 00:25:38-- http://yann.lecun.com/exdb/mnist/t10k-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: 1648877 (1.6M) [application/x-gzip]
Saving to: 't10k-images-idx3-ubyte.gz'

t10k-images-idx3-ub 100%[=====] 1.57M 8.98MB/s in 0.2s

2018-11-20 00:25:39 (8.98 MB/s) - 't10k-images-idx3-ubyte.gz' saved [1648877/1648877]

--2018-11-20 00:25:39-- http://yann.lecun.com/exdb/mnist/t10k-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: 4542 (4.4K) [application/x-gzip]
Saving to: 't10k-labels-idx1-ubyte.gz'

t10k-labels-idx1-ub 100%[=====] 4.44K --KB/s in 0s

2018-11-20 00:25:39 (543 MB/s) - 't10k-labels-idx1-ubyte.gz' saved [4542/4542]

ubuntu@ml2-thug:~/Deep-Learning/Caffe_/group1exam$

```

*Figure 2: output of running get\_mnist.sh*

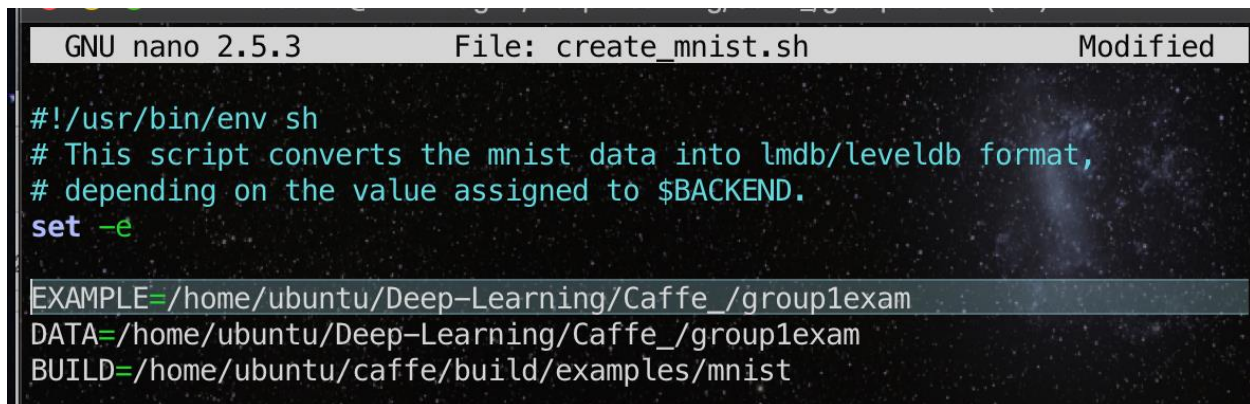
**Question 3: Open up the create\_mnist.sh and change the path to of EXAMPLE and DATA to the directory that you are working with. Then run the following the shell script.**

In the terminal, we opened up the create\_mnist script using nano Linux command. We changed the path in three places. The EXAMPLE, DATA, and BUILD.



- EXAMPLE: we changed ajafari to ubuntu and Mini\_Project to our exam directory name.
- DATA: we changed ajafari to ubuntu and Mini\_Project to our exam directory name.
- BUILD: here we only changed ajafari to ubuntu.

Figure 3 below contains the path variable after the paths were altered.

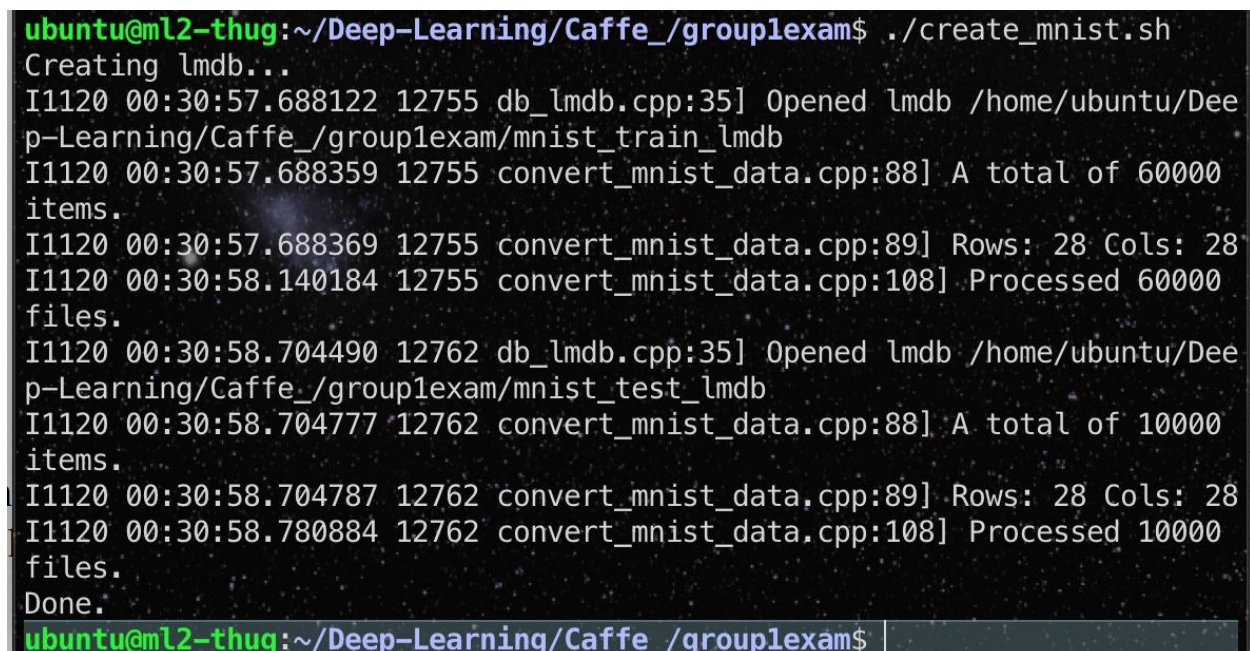


```
GNU nano 2.5.3      File: create_mnist.sh      Modified
#!/usr/bin/env sh
# This script converts the mnist data into lmdb/leveldb format,
# depending on the value assigned to $BACKEND.
set -e

EXAMPLE=/home/ubuntu/Deep-Learning/Caffe_/group1exam
DATA=/home/ubuntu/Deep-Learning/Caffe_/group1exam
BUILD=/home/ubuntu/caffe/build/examples/mnist
```

*Figure 3: Path changed in the create\_mnist.sh script*

Figure 4 has the output from running the create mnist script.

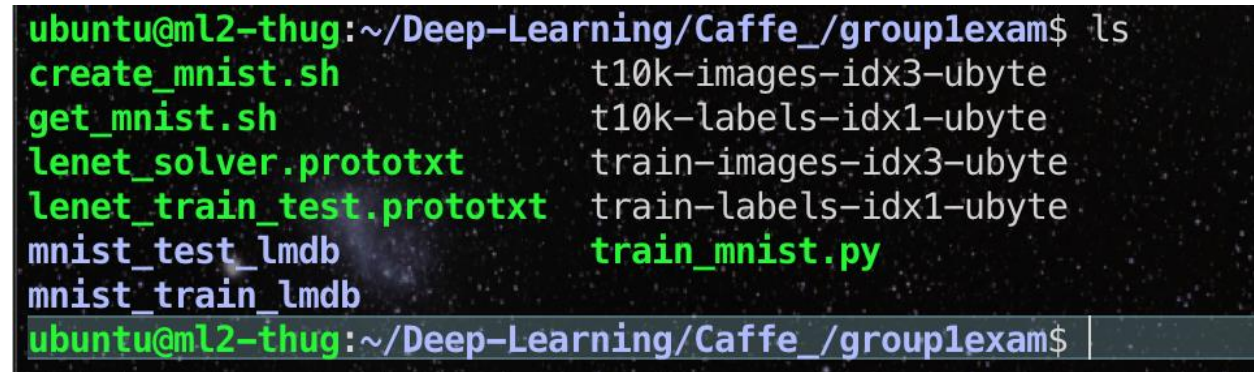


```
ubuntu@ml2-thug:~/Deep-Learning/Caffe_/group1exam$ ./create_mnist.sh
Creating lmdb...
I1120 00:30:57.688122 12755 db_lmdb.cpp:35] Opened lmdb /home/ubuntu/Deep-Learning/Caffe_/group1exam/mnist_train_lmdb
I1120 00:30:57.688359 12755 convert_mnist_data.cpp:88] A total of 60000 items.
I1120 00:30:57.688369 12755 convert_mnist_data.cpp:89] Rows: 28 Cols: 28
I1120 00:30:58.140184 12755 convert_mnist_data.cpp:108] Processed 60000 files.
I1120 00:30:58.704490 12762 db_lmdb.cpp:35] Opened lmdb /home/ubuntu/Deep-Learning/Caffe_/group1exam/mnist_test_lmdb
I1120 00:30:58.704777 12762 convert_mnist_data.cpp:88] A total of 10000 items.
I1120 00:30:58.704787 12762 convert_mnist_data.cpp:89] Rows: 28 Cols: 28
I1120 00:30:58.780884 12762 convert_mnist_data.cpp:108] Processed 10000 files.
Done.
ubuntu@ml2-thug:~/Deep-Learning/Caffe_/group1exam$
```

*Figure 4: create\_mnist.sh output in terminal.*

**Question 4: Check if the train and test lmdb files of the mnist data set are in your working directory.**

To check if the train and test lmdb files of the mnist data set are in our directory. We simply used a Linux command to list all the files in the directory. In figure 5 below, you can clearly see that our exam directory includes both train and test mnist data sets.

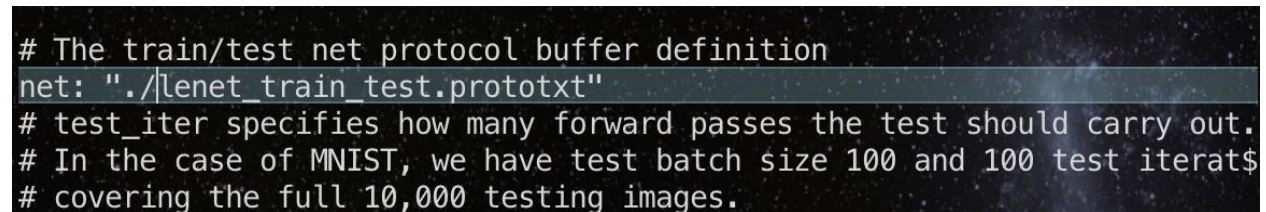
A terminal window with a dark background and light-colored text. The prompt is 'ubuntu@ml2-thug:~/Deep-Learning/Caffe\_/group1exam\$'. The command 'ls' has been executed, and the output is listed in two columns. The first column contains: 'create\_mnist.sh', 'get\_mnist.sh', 'lenet\_solver.prototxt', 'lenet\_train\_test.prototxt', 'mnist\_test\_lmdb', 'mnist\_train\_lmdb'. The second column contains: 't10k-images-idx3-ubyte', 't10k-labels-idx1-ubyte', 'train-images-idx3-ubyte', 'train-labels-idx1-ubyte', 'train\_mnist.py'. The prompt is now 'ubuntu@ml2-thug:~/Deep-Learning/Caffe\_/group1exam\$' again.

```
ubuntu@ml2-thug:~/Deep-Learning/Caffe_/group1exam$ ls
create_mnist.sh          t10k-images-idx3-ubyte
get_mnist.sh             t10k-labels-idx1-ubyte
lenet_solver.prototxt    train-images-idx3-ubyte
lenet_train_test.prototxt train-labels-idx1-ubyte
mnist_test_lmdb          train_mnist.py
mnist_train_lmdb
ubuntu@ml2-thug:~/Deep-Learning/Caffe_/group1exam$
```

*Figure 5: List of files in exam directory*

**Question 5: Download lenet solver.prototxt, lenet train test prototxt from GitHub and put them in the directory. Change the path of net in lenet solver.prototxt to your working path directory.**

Both prototxt files were included in the files gathered in question 1. Changing the directory in *net* in the lenet\_solver.prototxt is not required because the lenet\_train\_test.prototxt is already in the working directory. But for the sake of changing something we used the current directory symbol that can be seen in figure 6 below right before the lenet\_train\_test.prototxt

A terminal window showing a snippet of a .prototxt file. The text is as follows: '# The train/test net protocol buffer definition', 'net: \"./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 iterat\$', '# covering the full 10,000 testing images.'.

```
# The train/test net protocol buffer definition
net: \"./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 iterat$
# covering the full 10,000 testing images.
```

*Figure 6: Partial screenshot of the lenet\_solver.prototxt file*

**Question 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.**

After following all the steps from questions 1 through 5, the `train_mnist.py` python file ran successfully. We changed the iteration to 2000 for the program to run faster. Five figures printed out. The loss decreased, as it should, but then it started to increase again. Even though the loss increases again, the accuracy was 100 percent. Figure 7 holds a screenshot of the accuracy and loss for the last iteration. We did not use the `my_root` variable. The variable is used to change the directory; however, we do not need to change the directory because we are running the files in the working directory. We left the variable commented.

```
I1120 14:59:24.100379 2768 solver.cpp:239] Iteration 1900 (222.667 iter/s, 0.449102s/100 iters), loss = 0.127615
I1120 14:59:24.100461 2768 solver.cpp:258] Train net output #0: loss = 0.127615 (* 1 = 0.127615 loss)
I1120 14:59:24.100471 2768 sgd_solver.cpp:112] Iteration 1900, lr = 0.00877687
Iteration 1900 testing... accuracy: 1.0
```

*Figure 7: Partial output from `train_mnist.py` file*

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

The question is asking what kernel would be best for certain digits in the mnist dataset. Initially, we approached this question by investigating what effects does changing the kernel have on accuracy and loss. Using iteration 2000, we check to see these effects in three different ways:

1. Table 1: Change the kernel size just for convolution layer 1
2. Table 2: Change the kernel size just for convolution layer 2
3. Table 3: Change the kernel sizes for both convolution layers.

*Table 1. Changing Kernel sizes in FIRST conv layer*

KER_C1	STR_C1	KER_C2	STR_C2	ACC	LOSS
2	1	5	1	1.0	0.116796
4	1	5	1	1.0	0.122505
6	1	5	1	1.0	0.131108
8	1	5	1	1.0	0.137875

10	1	5	1	1.0	0.15693
----	---	---	---	-----	---------

*Table 2: Changing kernel size in SECOND conv layer*

KER_C1	STR_C1	KER_C2	STR_C2	ACC	LOSS
5	1	2	1	0.99	0.0989604
5	1	4	1	1	0.12716
5	1	6	1	1	0.139601
5	1	8	1	0.99	0.117865
5	1	10	1	0.99	0.124134

*Table 3: Changing kernel sizes in BOTH conv layers*

KER_C1	STR_C1	KER_C2	STR_C2	ACC	LOSS
2	1	2	1	0.98	0.133298
4	1	4	1	0.99	0.097727
6	1	6	1	1.0	0.131091
8	1	8	1	0.98	0.125921
10	1	10	1	0.98	0.135465

The results were inconclusive. We did not observe any significant changes in accuracy and loss. We changed kernel sizes (sizes 2-10) between models, but within each model, we had the same kernel sizes. Figure 8 shows the outputs of the kernels of both convolution layers along with the image effected by it. By plainly looking at the initialized kernels, initially, it's hard to spot any patterns. But when we increase the sizes, in each of the images (1st layer & 2nd of the images) we can see some sort of pattern forming. However, without looking at the gradients, it's not really possible to identify kernels that would be useful for particular numerals. As requested by the professor, we've attached the excel file in the code folder.



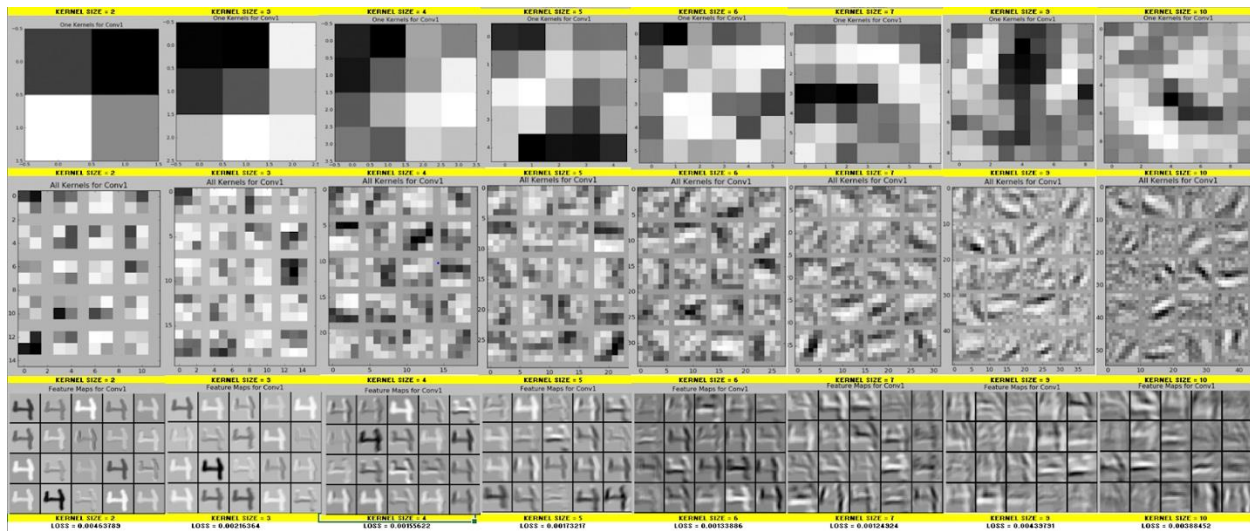


Figure 8: Kernel 2 --> kernel size 10

**Question 8: How does the performance of the convolution network compare with the multilayer networks that you used in Exam#1?**

In both exams, we worked with 28x28 pixel images. In exam 1, we utilized multilayer networks. In one test, we had 4 epochs (iterations) and a batch size of 128. This test took 36 seconds to run. For exam 2, we used CNN with caffe. In one test, we used 2000 iterations which is 500 times the amount of iterations used in exam 1. Logically one would think it would take longer for 2000 iterations to run, however, the process ran quicker than it did for exam 1 with a run time of 28 seconds. In conclusion, convolution neural networks perform faster than multilayer networks. The table below shows the results from similar test in exam 1 and 2

Table 4: Comparison between exam 1 and 2

EXAM	Iterations	Batch Size	Runtime(seconds)
Exam#1	4	128	35.876
Exam#2	2000	100	27.886

**Question 9: Change the size of the batch size parameter. 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.**

We decided to use three different number of iterations. We tested the network stochastically for iteration 2000 and 10000. Table 5 below presents the results from the test. Both were

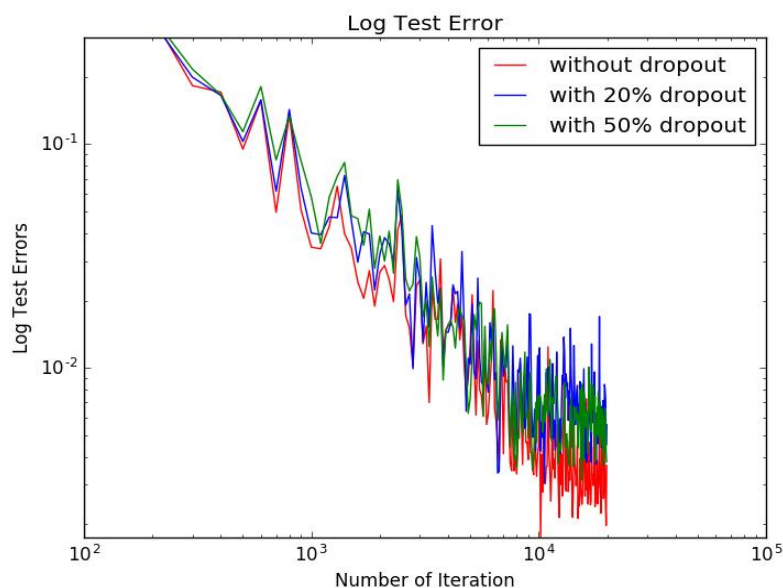
significantly faster than mini and total batch. One this advantage of making the batch size “very large” in a convolution with caffe is the computational time. It increases by a lot.

**Table 5: testing runtime with different batch sizes**

Number of Iterations	Batch Size	Runtime (seconds)
20000	64	111.369
20000	256	229.137
2000	1	24.220
2000	100	27.886
2000	10000	605.729
10000	1	50.284
10000	1000	625.090

**Question 10: Use a dropout layer at layer fc1. Make fc1 the top and the bottom for the dropout Layer. Does dropout improve the testing error?**

We initially added a new dropout layer to the currently existing network architecture & we found that there wasn't a significant change in the (log) error, i.e the model was not overfitting. This can be seen in figure 9.

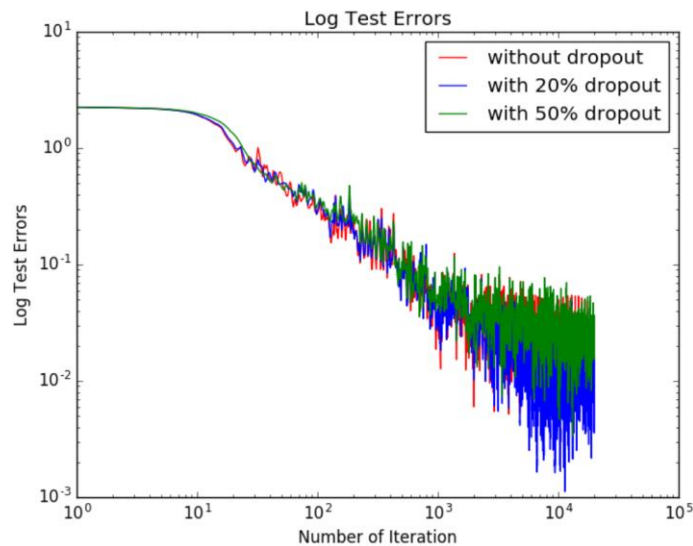


*Figure 9: Test error with and without dropout nodes.*



**Question 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 the network. Describe how the performance changes as the number of layers increases – both in terms of training time and performance.**

We added 2 more layers to the network. Conv3 & pool3. However, in order to account for the size issues that we were facing we added a padding of 1 to the pool2 & reduced the kernel size to 3. We also added dropouts to the network & tracked the performance of the model with and without. As we can see, with dropouts, the model performed better (slightly). i.e, we are overfitting the model by adding new unnecessary layers & the dropout (20%) shows us that the error of the model decreased (on a log scale) and the model performs better with dropouts.



**Question 12: Try one other training function from the list on this page compare the performance with gradient descent.**

I used the Adam optimizer and the stochastic gradient descent. Table 6 below contains the results from using the different optimizer. Our assumption was that there shouldn't be much of a difference; however, Adam was 3 time slower than SGD. The ADAM loss is slightly better than SGD.

*Table 6: testing with different optimizers*

Solver	Runtime (Seconds)	Accuracy	Loss
SGD	43.497	1.0	0.129293
ADAM	136.065	0.99	0.011642