Homework-3 Report

MatConvNet –

Algorithm:

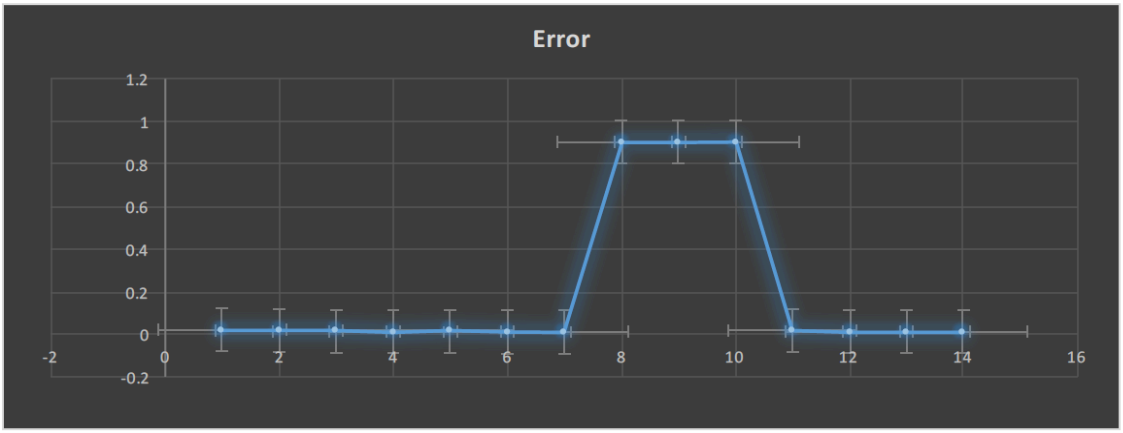
1. Download cnn_mnist_6156.m and run the original
2. Change Topology of Neural Network [Adding & removing pair of convolutional and max- pooling layers]
3. Add Dropout Layer to Neural Network [Change the dropout rate between 0.5 to 1]
4. Add Relu layer after pairs of convolutional and max- pooling layers
5. Compare the result

Discussion & Results:

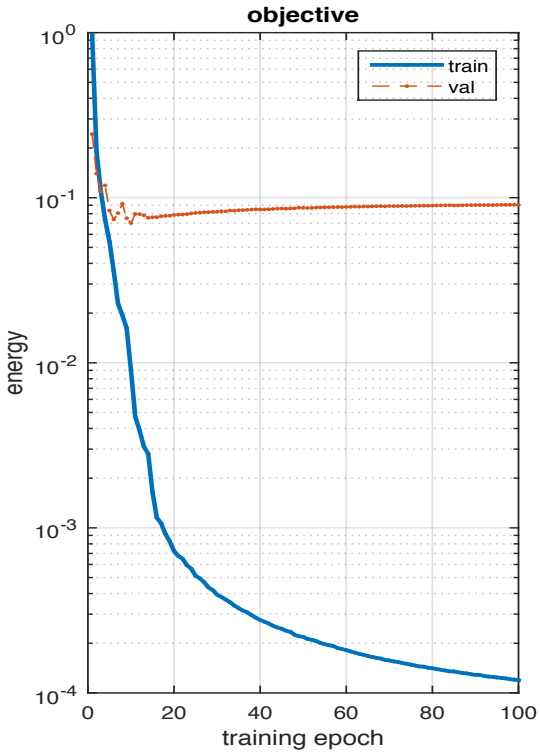
- Adding dropout layer to the original LeNet decreased the error.
- Lowest error was calculated for dropout rate (0.75).
- Dropout layer decreases error due to variance as it randomly removes nodes from input & hidden layers in order to avoid over fitting.
- Decreasing the Number of layers from Original LeNet increased the error.
- **Lowest Error [0.099] was recorded for 6th Experiment in which Relu and Dropout Layer [dropout rate 0.75] was added to Original LeNet**

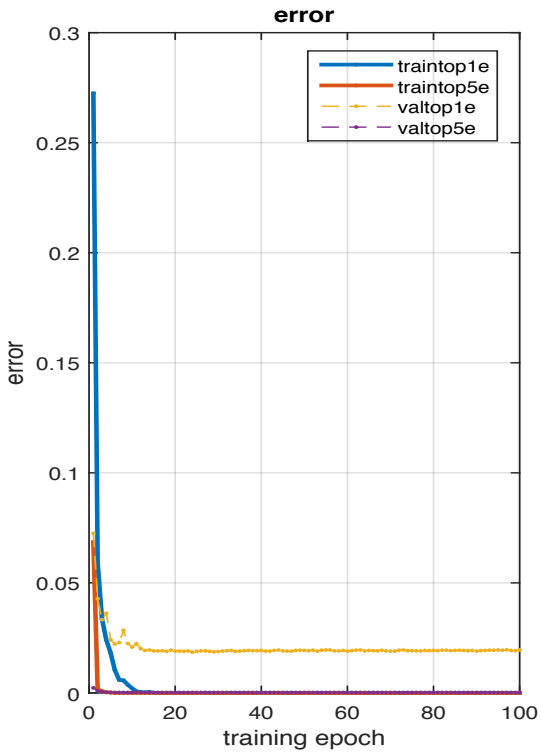
Convolution Neural Network

Experiment No	Dropout	Feature	Combination	Error
1	No	Original LeNet	Conv 5 Po 2 Conv 5 Po 2 Conv 4 Relu Conv 1 SL	0.0195
2	No	Original LeNet + Relu	Conv 5 Po 2 Relu Conv 5 Po 2 Conv 4 Relu Conv 1 SL	0.0162
3	Yes	Original LeNet + droput_0.5	Conv 5 Po 2 Conv 5 Po 2 Conv 4 Relu Dp Conv 1 SL	0.0136
4	Yes	Original LeNet + droput_0.75	Conv 5 Po 2 Conv 5 Po 2 Conv 4 Relu Dp Conv 1 SL	0.0111
5	Yes	Original LeNet + droput_0.85	Conv 5 Po 2 Conv 5 Po 2 Conv 4 Relu Dp Conv 1 SL	0.0134
6	**Yes**	**Original LeNet + Relu + droput_0.75**	**Conv 5 Po 2 Relu Conv 5 Po 2 Conv 4 Relu Dp Conv 1 SL**	**0.0099**
7	**Yes**	**Original LeNet + Relu + droput_0.75**	**Conv 5 Relu Po 2 Conv 5 Po 2 Conv 4 Relu Dp Conv 1 SL**	**0.0101**
8	Yes	1st Layer + Pool (6 Layer)	Conv 27 Po 2 Relu Dp Conv 1 SL	0.902
9	Yes	1st Layer (5 Layer)	Conv 28 Relu Dp Conv 1 SL	0.902
10	Yes	2 Layer (8 Layer)	Conv 13 Po 2 Conv 7 Po 2 Relu Dp Conv 1 SL	0.903
11	Yes	9 Layer	Conv 9 Po 2 Conv 7 Po 2 Conv 2 Relu Dp Conv 1 SL	0.0154
12	Yes	11 Layer	Conv 9 Po 2 Relu Conv 7 Po 2 Relu Conv 2 Relu Dp Conv 1 SL	0.0122
13	Yes	10 Layer	Conv 9 Po 2 Relu Conv 7 Po 2 Conv 2 Relu Dp Conv 1 SL	0.0117
14	Yes	Original LeNet + Relu + droput_0.75	Conv 5 Po 2 Conv 5 Po 2 Relu Conv 4 Relu Dp Conv 1 SL	0.0115

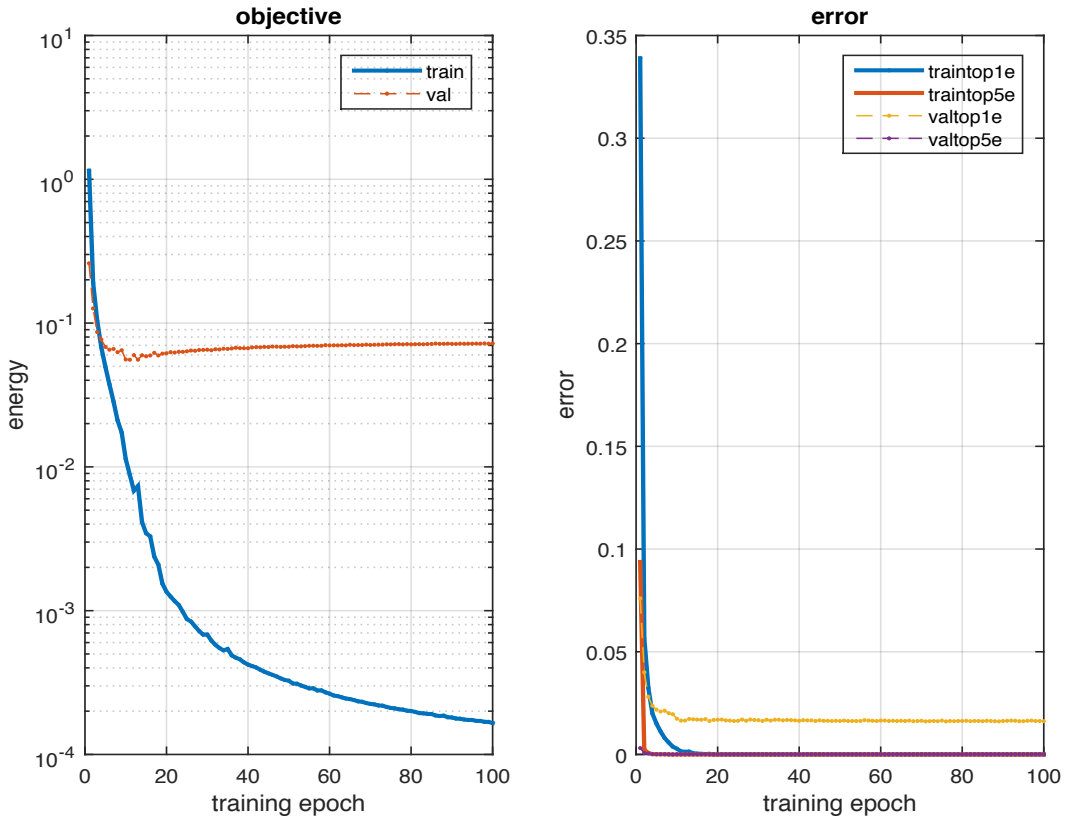


Following are the result graph for above mentioned experiment:
Experiment No 1: [Error: 0.0195]
[Conv 5 Po 2 Conv 5 Po 2 Conv 4 Relu Conv 1 SL]

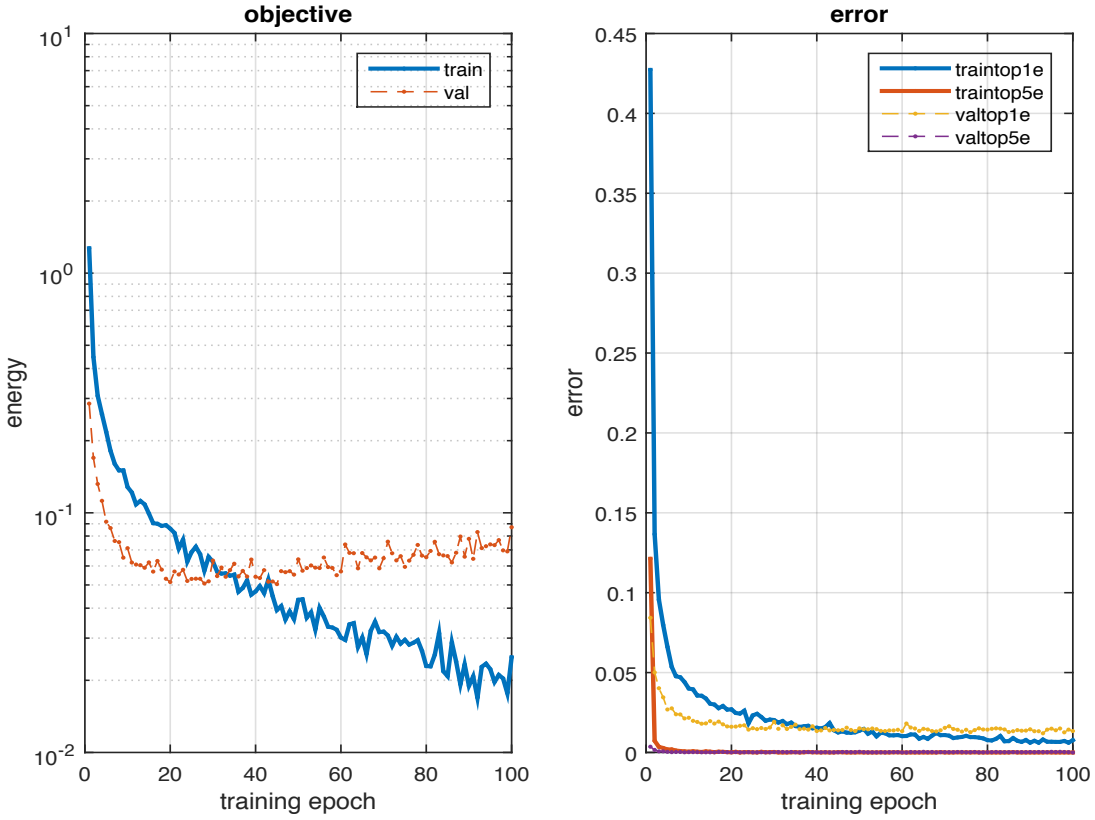




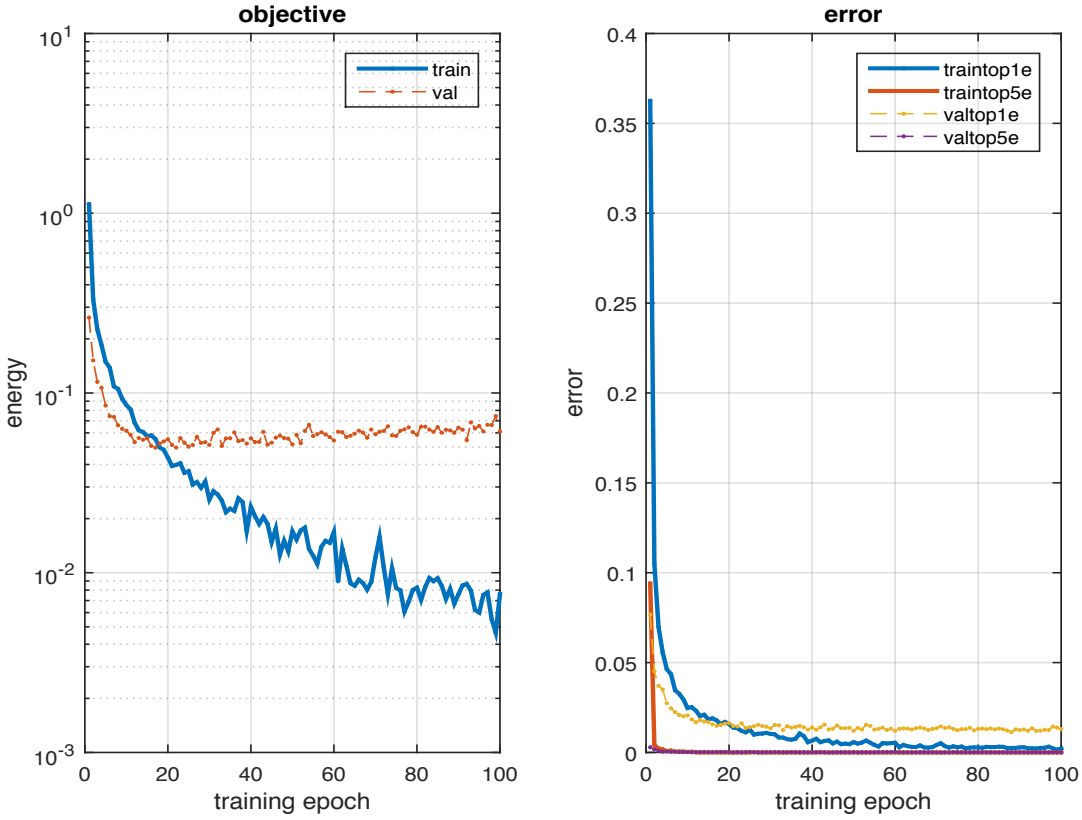
Experiment No 2: [Error: 0.0162]
[Conv 5 Po 2 Relu Conv 5 Po 2 Conv 4 Relu Conv 1 SL]



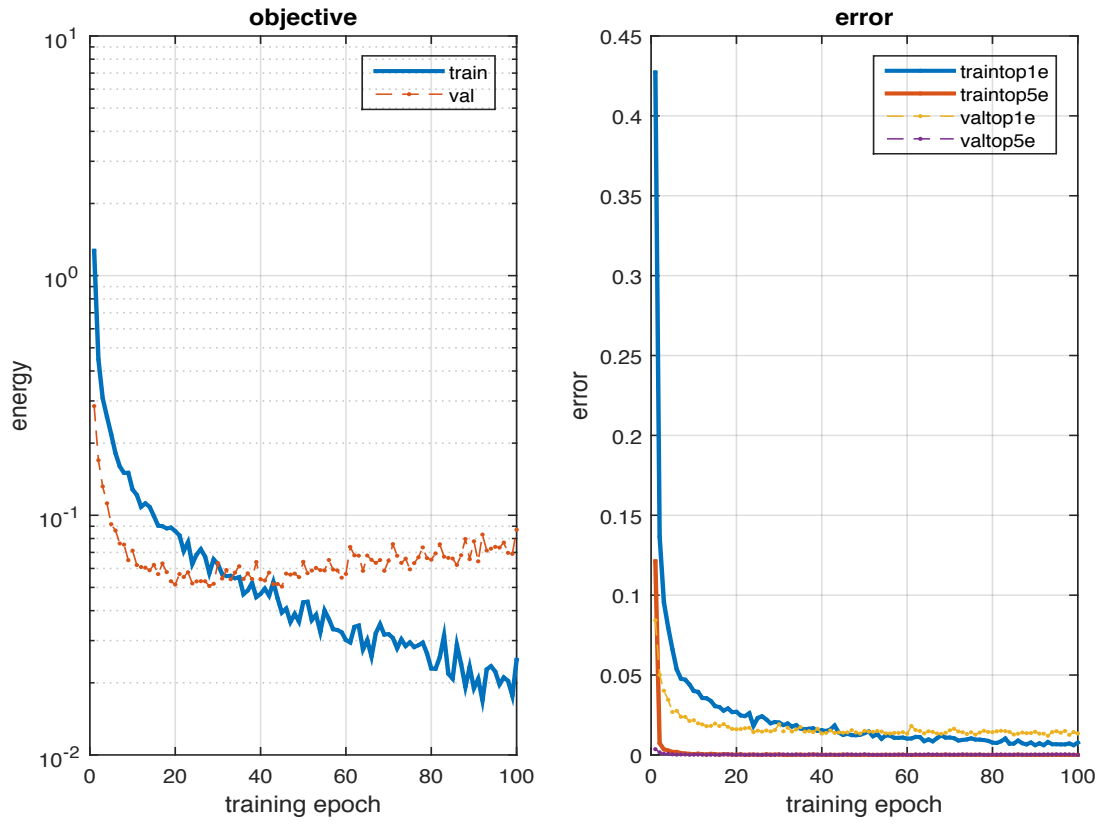
Experiment No 3: [Error: 0.0136]
[Conv 5 Po 2 Conv 5 Po 2 Conv 4 Relu Dp Conv 1 SL]



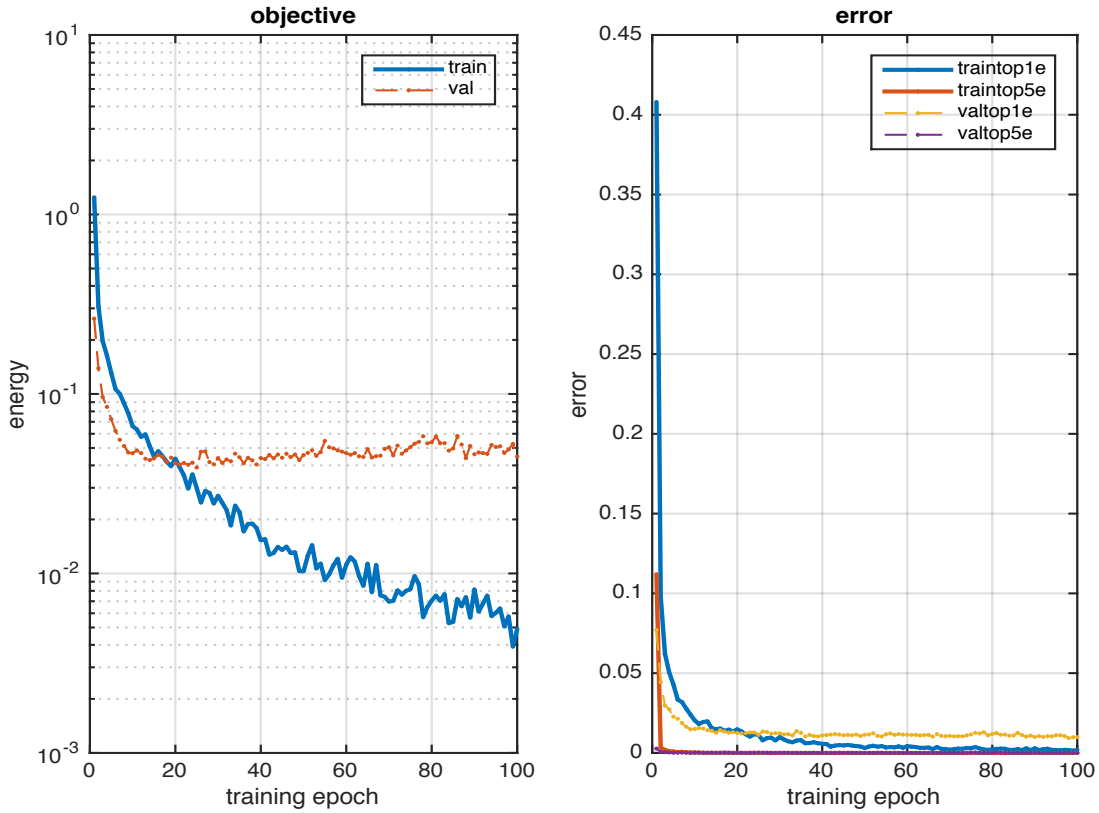
Experiment No 4: [Error: 0.0111]
[Conv 5 Po 2 Conv 5 Po 2 Conv 4 Relu Dp Conv 1 SL]



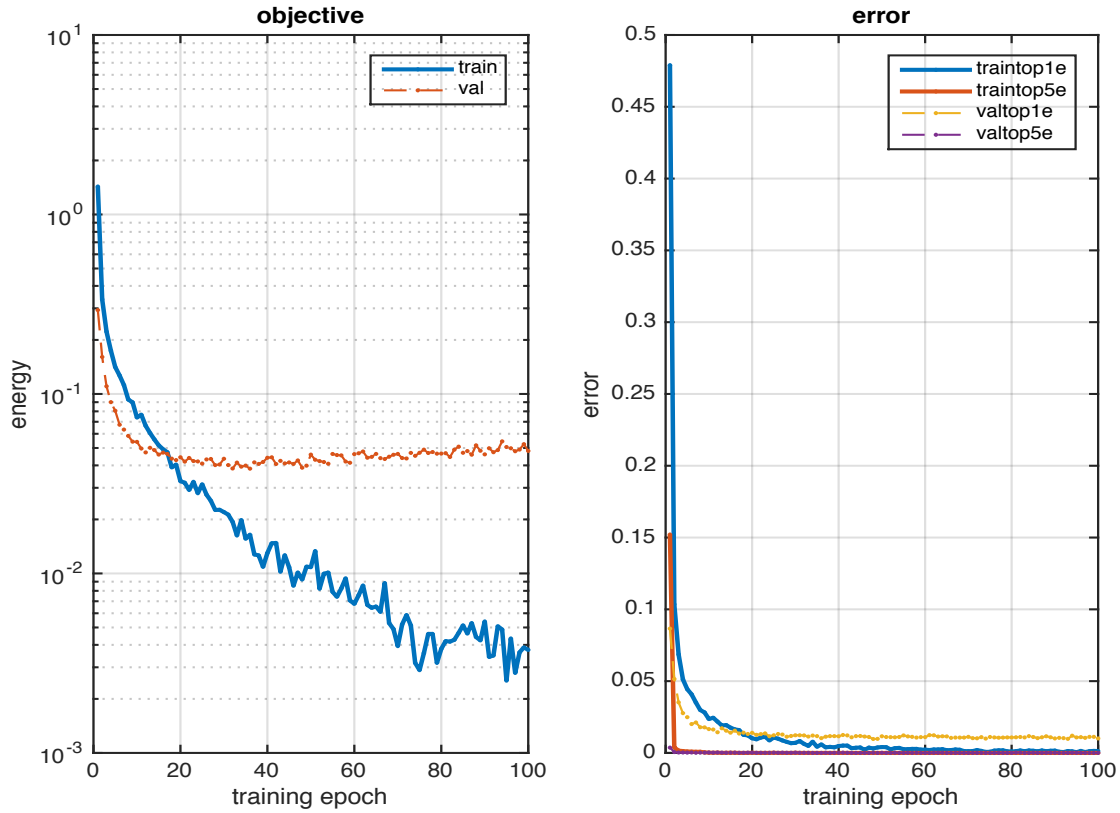
Experiment No 5: [Error: 0.0134]
[Conv 5 Po 2 Conv 5 Po 2 Conv 4 Relu Dp Conv 1 SL]



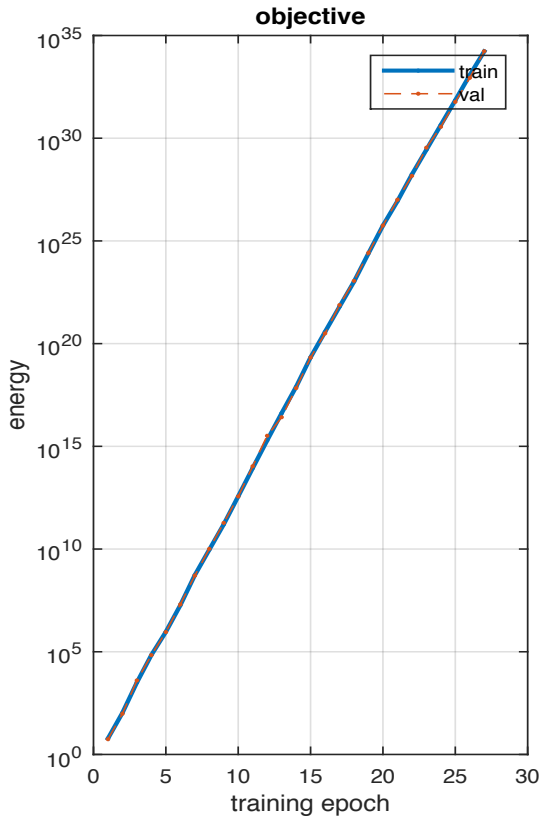
Experiment No 6: [Error: 0.0099]
[Conv 5 Po 2 Relu Conv 5 Po 2 Conv 4 Relu Dp Conv 1 SL]

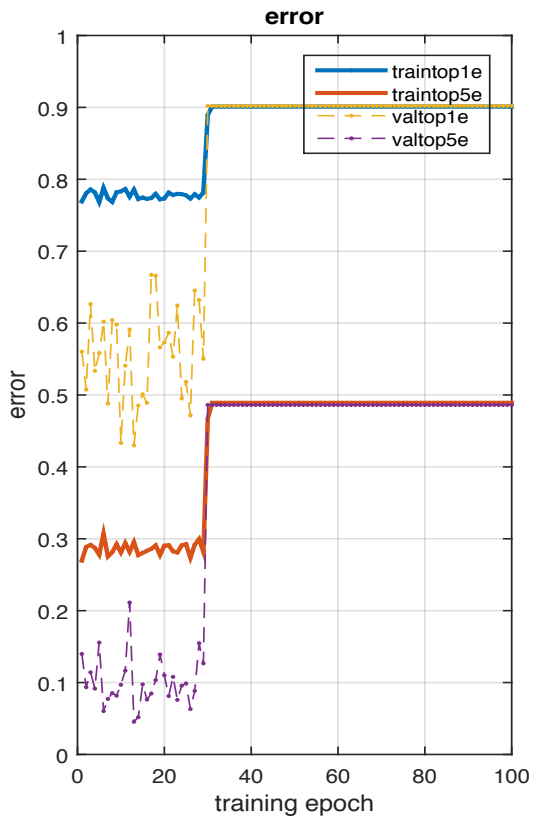


Experiment No 7: [Error: 0.0101]
[Conv 5 Relu Po 2 Conv 5 Po 2 Conv 4 Relu Dp Conv 1 SL]

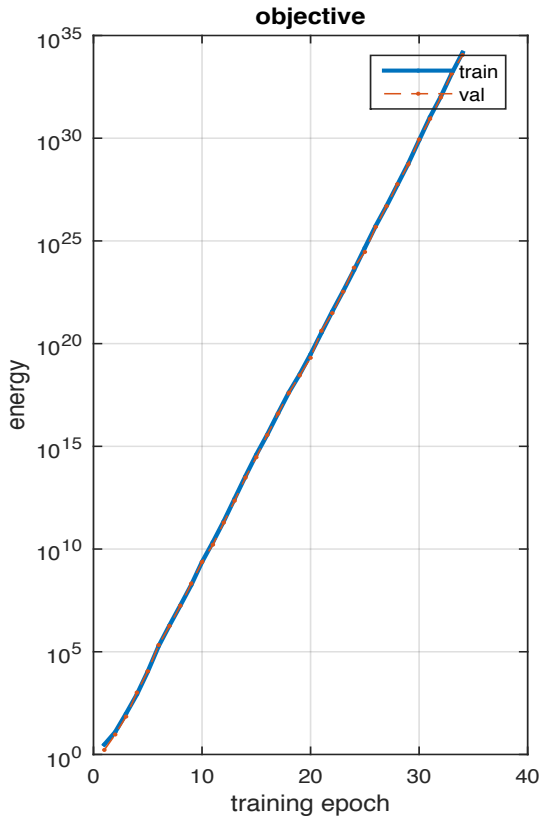


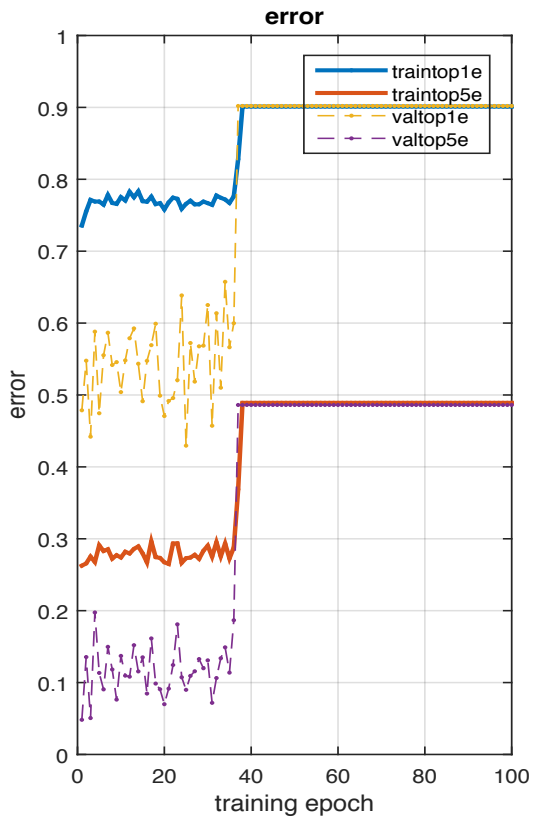
Experiment No 8: [Error: 0.902]
[Conv 27 Po 2 Relu Dp Conv 1 SL]



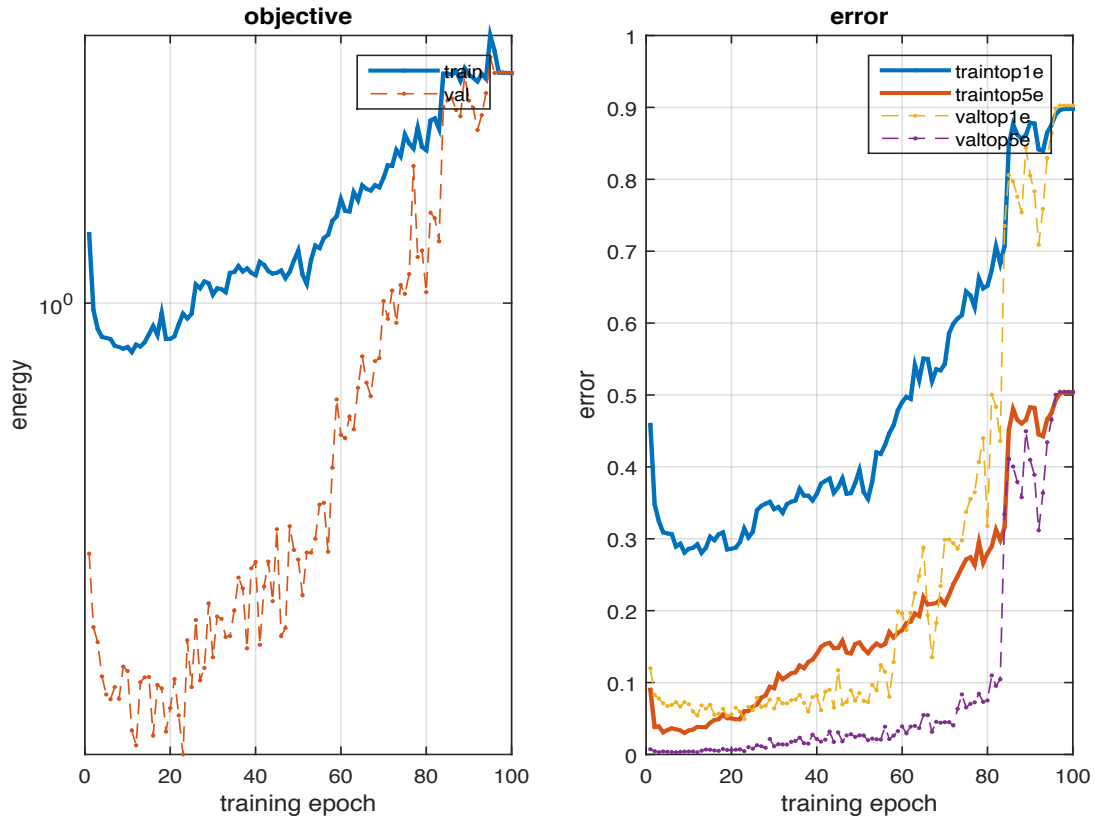


Experiment No 9: [Error: 0.902]
[Conv 28 Relu Dp Conv 1 SL]

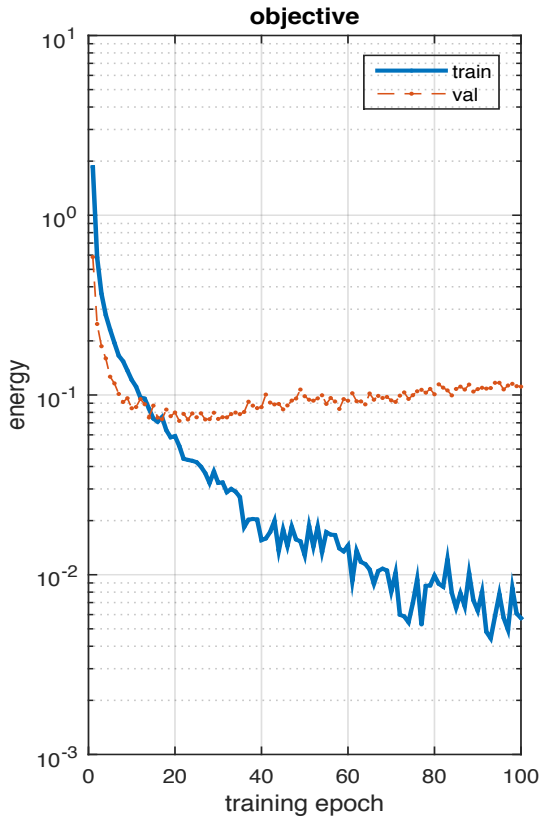


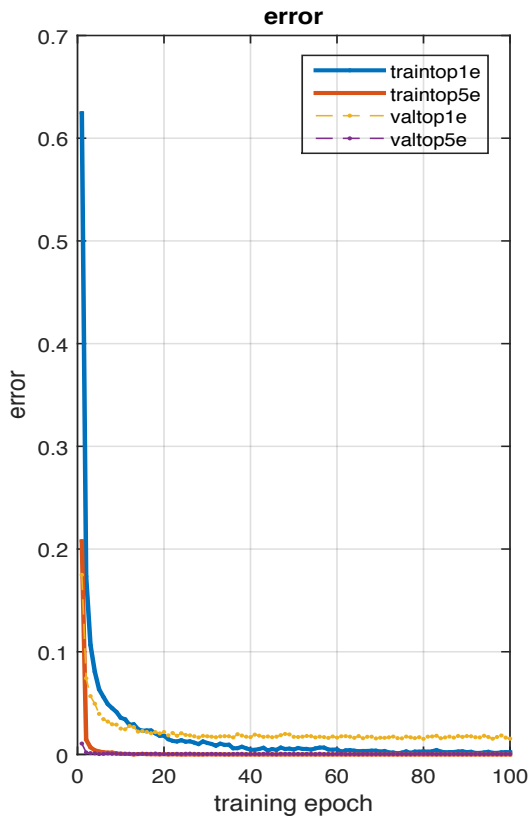


Experiment No 10: [Error: 0.903]
[Conv 13 Po 2 Conv 7 Po 2 Relu Dp Conv 1 SL]

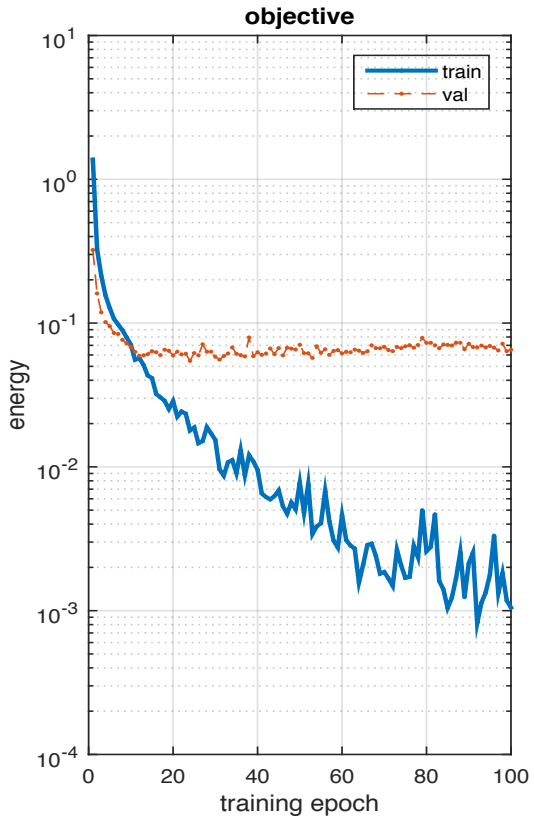


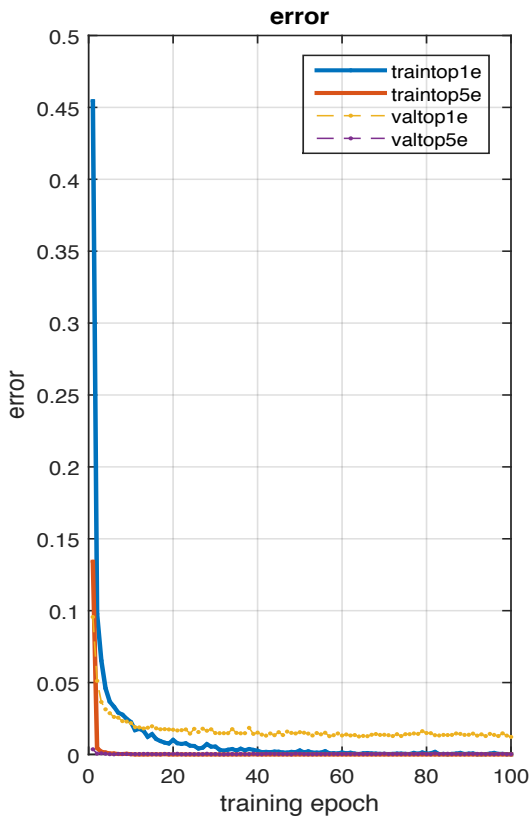
Experiment No 11: [Error: 0.0154]
[Conv 9 Po 2 Conv 7 Po 2 Conv 2 Relu Dp Conv 1 SL]



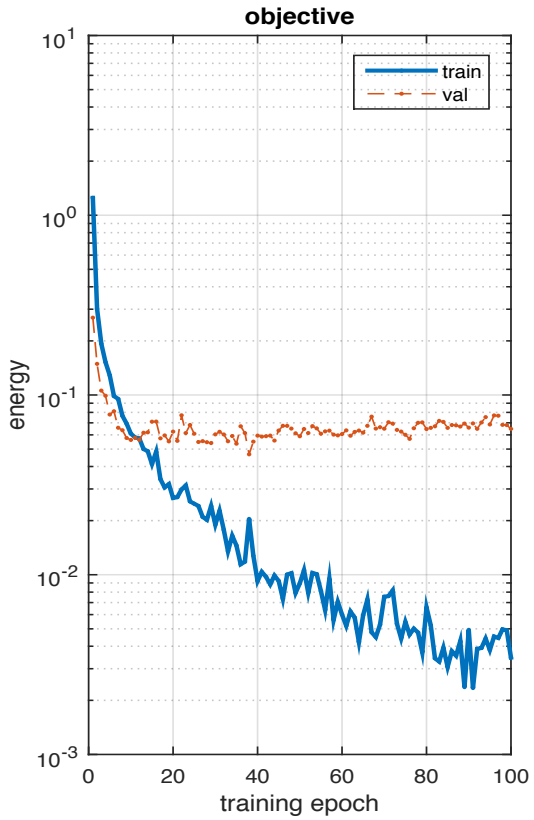


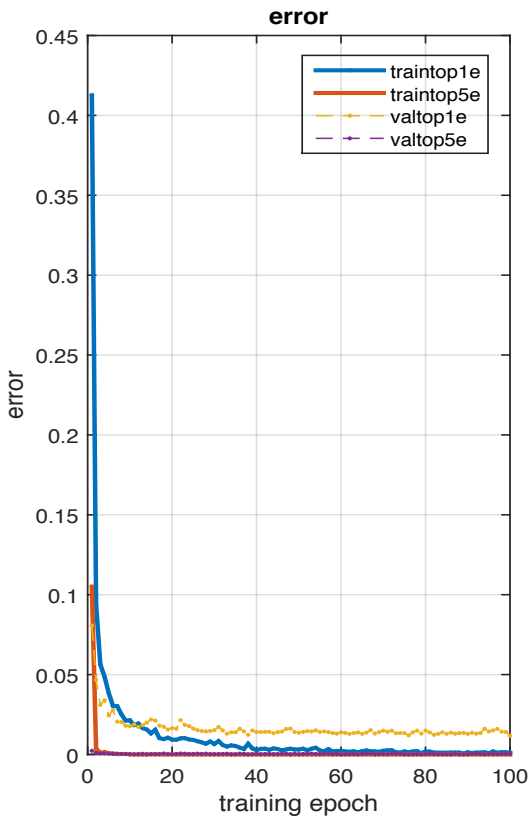
Experiment No 12: [Error: 0.0122]
[Conv 9 Po 2 Relu Conv 7 Po 2 Relu Conv 2 Relu Dp Conv 1 SL]



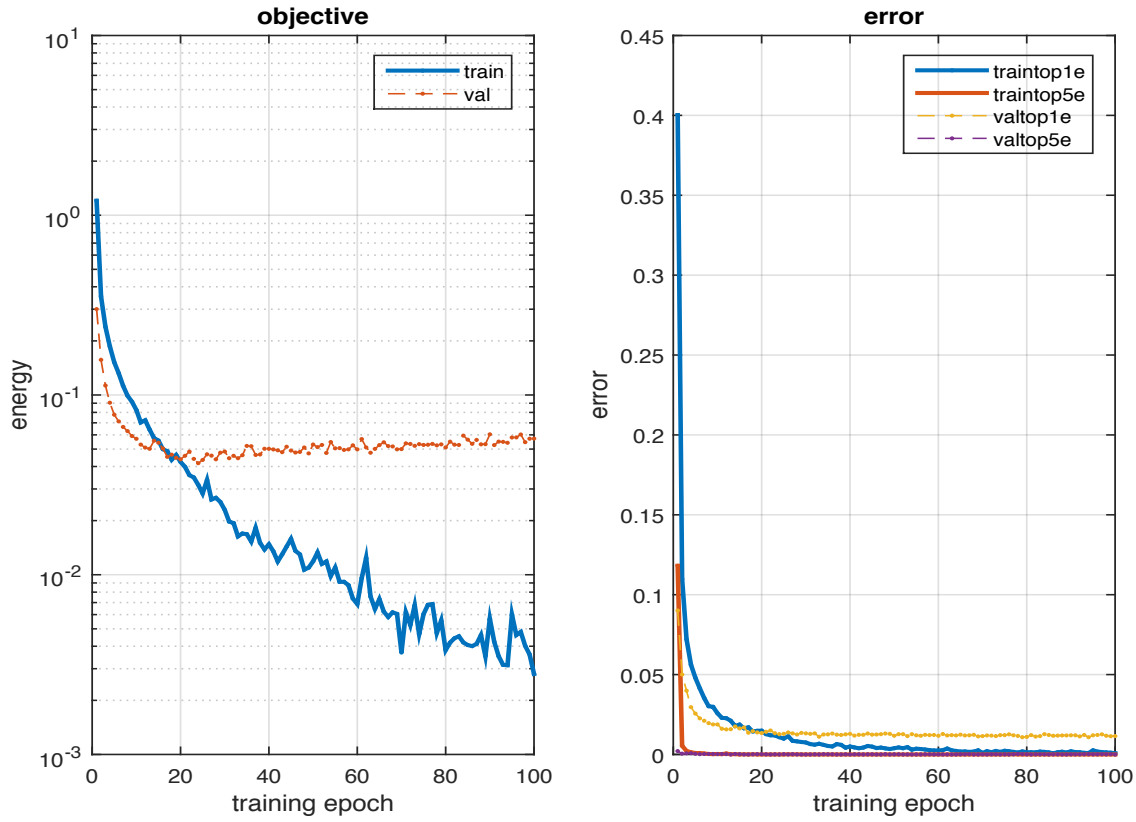


Experiment No 13: [Error: 0.0117]
[Conv 9 Po 2 Relu Conv 7 Po 2 Conv 2 Relu Dp Conv 1 SL]





Experiment No 14: [Error: 0.0115]
[Conv 5 Po 2 Conv 5 Po 2 Relu Conv 4 Relu Dp Conv 1 SL]



LibSVM -

Algorithm:

1. Load the data from imdb.mat file which is created during execution of cnn_mnist_6156.m file
2. Transform image.data from 4 Dimensional to 2 Dimensional (784 X 20000) and store it in X
3. Take transpose of X making it X [20000,784]
4. Divide X by 255 to scale the data
5. Create a sparse Matrix out of X
6. Load images.labels into Y and make a Transpose of Y
7. Divide data in X by 255 to scale data to value between [0 to 1]
8. Create sparse matrix from X as it contains lot of rows with value 0
9. Use libsvmwrite, libsvmread functions to write & read data
10. Take first 10000 rows as train data & train label from X & Y
11. Take last 10000 rows as test data & test label from X & Y
12. Run Cross validation function on each type of Kernel to get the best cost (-c) and gamma (-g) value.
13. Use this cost and gamma values to train the model and then use it on test data to predict accuracy.

Results & Discussion:

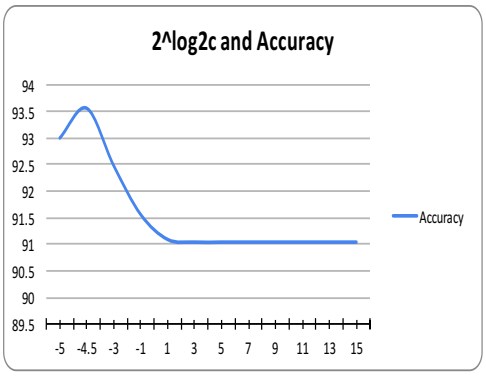
- First run cross validation function for **–c (Range: -5 to 15) and –g (Range: -15 to 3)** used the best value of –c & -g on test data.
- Increasing the value of –c increased training time for svm
- Result of cross validation function are show in image a presented below.
- **Radial Basis Kernel gave highest accuracy of 96.76 % on test data among all kernel.**

1. **Linear Kernel**: Highest Accuracy on test data: **93.58% [-c 0.05]**
2. **Polynomial Kernel Degree 2:** Highest Accuracy on test data: **96.59% [-c 8192 –g 0.00048 –r 0]**
3. **Polynomial Kernel Degree 4:** Highest Accuracy on test data: **96.52% [-c 0.5 –g 0.125 –r 1]**
4. **Radial Basis Kernel:** Highest Accuracy on test data: **96.76 % [-c 8 –g 0.5]**

Following is the result of Cross Validation Function & Test Data on given kernels

Cross Validation Function

Linear	2^log2c	Accuracy
	-5	93
	-4.5	**93.56**
	-3	92.48
	-1	91.56
	1	91.09
	3	91.04
	5	91.04
	7	91.04
	9	91.04
	11	91.04
	13	91.04
	15	91.04



Result on Train Data

Kernel: Linear Higest Accuracy on Test Data		
-c	cost	0.05
Accuracy		93.58%

Poly 2	2^log2c / 2^log2g	-15	-13	-11	-9	-7	-5	-3	-1	1	3
-r (coef0) : 0	-5	11.62	11.62	11.62	11.62	78.13	93.25	96.05	96.05	96.05	96.05
	-3	11.62	11.62	11.62	34.29	89.08	95.15	96.05	96.05	96.05	96.05
	-1	11.62	11.62	11.62	78.13	93.25	96.09	96.05	96.05	96.05	96.05
	1	11.62	11.62	11.62	78.13	93.25	96.09	96.05	96.05	96.05	96.05
	3	11.62	11.62	34.29	89.08	95.15	96.09	96.05	96.05	96.05	96.05
	5	11.62	34.29	89.09	95.15	96.09	96.05	96.05	96.05	96.05	96.05
	7	11.62	78.13	93.25	96.09	96.05	96.05	96.05	96.05	96.05	96.05
	9	34.29	89.08	95.15	96.09	96.05	96.05	96.05	96.05	96.05	96.05
	11	78.13	93.25	96.09	96.05	96.05	96.05	96.05	96.05	96.05	96.05
	13	89.08	95.15	**96.1**	96.05	96.05	96.05	96.05	96.05	96.05	96.05
	15	11.62	96.09	96.05	96.05	96.05	96.05	96.05	96.05	96.05	96.05

Kernel: Polynomial Degree: 2 Higest Accuracy on Test Data		
-c	cost	8192
-g	gamma	0.00048
-r	coeff0	0
Accuracy		96.59 %

Poly 4	2^log2c / 2^log2g	-15	-13	-11	-9	-7	-5	-3	-1	1	3
-r (coef0) : 0	-5	11.49	11.49	11.49	11.49	34.44	92.29	93.63	93.65	93.65	93.65
	-3	11.49	11.49	11.49	11.49	62.09	93.79	93.65	93.65	93.65	93.65
	-1	11.49	11.49	11.49	11.71	79.21	93.32	**93.92**	93.65	93.65	93.65
	1	11.49	11.49	11.49	15.87	88.46	93.71	93.65	93.65	93.65	93.65
	3	11.49	11.49	11.49	34.44	92.29	93.63	93.65	93.65	93.65	93.65
	5	11.49	11.49	11.49	62.09	93.79	93.65	93.65	93.65	93.65	93.65
	7	11.49	11.49	11.71	79.21	93.92	93.65	93.65	93.65	93.65	93.65
	9	11.49	11.49	15.87	88.46	93.71	93.65	93.65	93.65	93.65	93.65
	11	11.49	11.49	34.44	92.29	93.63	93.65	93.65	93.65	93.65	93.65
	13	11.49	11.49	62.09	93.79	93.65	93.65	93.65	93.65	93.65	93.65
	15	11.49	11.71	79.21	93.92	93.65	93.65	93.65	93.65	93.65	93.65

Kernel: Polynomial Degree: 4 Higest Accuracy on Test Data		
-c	cost	0.5
-g	gamma	0.125
-r	coeff0	1
Accuracy		96.52%

Radial	2^log2c / 2^log2g	-4	-3	-2	-1	0	1
	-1	93.74	47.7	20	18.09	11.62	11.62
	0	95.59	79.82	34.65	18.76	11.64	11.62
	1	95.77	81.11	38.82	19.12	11.62	11.62
	2	95.77	81.11	38.82	19.12	11.64	11.62
	3	**95.77**	81.11	38.82	19.12	11.64	11.62

Kernel: Radial basis Higest Accuracy on Test Data		
-c	cost	8
-g	gamma	0.5
Accuracy		96.76%