LibSVM Experiments

Algorithm

1. Reshape images.data to [728*20000] matrix
2. Take the 1st 10000 entries and apply sparse() function. This creates sparse training data matrix.
3. Take next 10000 entries and apply sparse() function. This creates sparse testing data matrix.
4. Pass sparse training data and labels from images.lables to libsvmwrite(). This generates training data file in svm format.
5. Pass sparse testing data and labels from images.lables to libsvmwrite(). This generates testing data file in svm format.
6. Scale testing and training data on the scale of 0-1 using the same scaling function.
7. Follow grid search approach using n-fold cross validation to find cost parameter that gives best accuracy for Linear SVM. Use that to train Linear SVM model and find accuracy on testing data
8. Follow a grid search approach and use n-fold cross validation to find cost and gamma that give best accuracy for Polynomial degree 2 SVM. Use those parameters to train Polynomial degree 2 SVM model and find accuracy on testing data.
9. Follow a grid search approach and use n-fold cross validation to find cost and gamma that give best accuracy for Polynomial degree 4 SVM. Use those parameters to train Polynomial degree 4 SVM model and find accuracy on testing data.
10. Follow grid search approach and use n-fold cross validation to find cost and gamma that gives best accuracy for RBF SVM. Use those parameters to train RBF SVM model and find accuracy on testing data.

Design decisions:

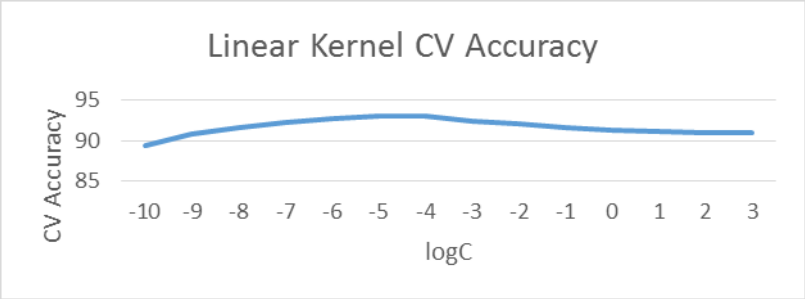
- I used the best parameter values suggested by cross-validation, to find accuracy on testing data and checked a few values above and below in the range of suggested parameters that gave best accuracy and stopped the search for better parameters when the accuracy started falling.
- For Polynomial kernel I did not include the 3rd parameter coef0 in grid search as it would take n^3 time to run for the 3 parameters. Instead I tried a few values in a range for coeff0 for the best cost and gamma values suggested by cross validation and chose the one for which the accuracy was highest.

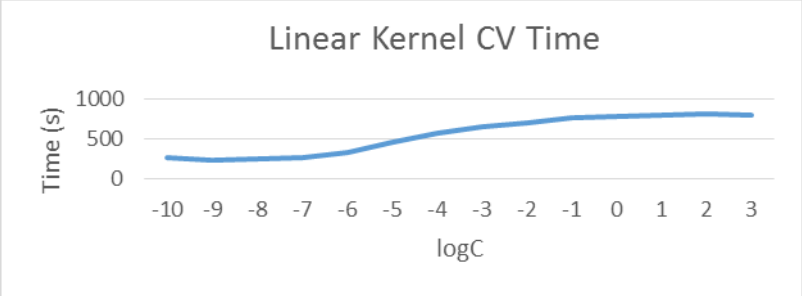
Files

- loadData.m – Creates mnist.train and mnist.test data.
- bestParamLinear.m - Uses grid search to find cost parameter that gives best accuracy using n-fold cross validation.
- bestParamPoly2.m - Uses grid search to find cost & gamma parameter that gives best accuracy using n-fold cross validation.
- bestParamPoly4.m - Uses grid search to find cost & gamma parameter that gives best accuracy using n-fold cross validation.
- bestParamRBF.m - Uses grid search to find cost & gamma parameter that gives best accuracy using n-fold cross validation.
- svm_mnist_6156.m - builds and tests the models for different kernels using the best parameters.
- mnist.train - libsvm format sparse matrix training data created from imdb.dat.
- mnist.test - libsvm format sparse matrix testing data created from imdb.dat.
- mnist_scale.train - Scaled to 0-1. This is used to train the model.
- mnist_scale.test - Scaled to 0-1. This is used to test the model.

Cross Validation Result for Linear Kernel

logC	CV Accuracy	Time (s)
-10	89.37	263.014
-9	90.83	228.471
-8	91.53	246.198
-7	92.24	269.724
-6	92.73	332.273
-5	92.99	456.577
-4	92.96	564.704
-3	92.41	648.02
-2	92.13	704.319
-1	91.59	767.691
0	91.28	783.119
1	91.1	804.606
2	90.99	807.854
3	90.99	802.368





Results on Actual Training and Testing data

Using logC= -5 i.e. C= 0.03125
Accuracy= 93.15%
Training Time= 17.504450s
Testing Time= 28.533582s

On trying a few values near the CV results, the highest accuracy result for C= 0.05 is
Accuracy= 93.2%
Training Time= 16.231007s
Testing Time= 27.544710s

Cross Validation Result for RBF Kernel

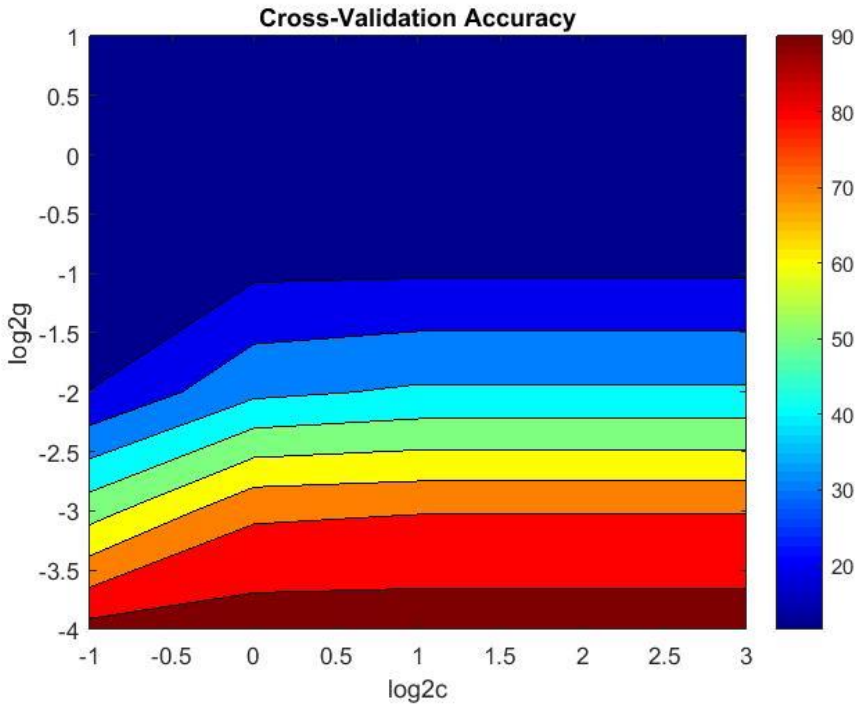
logC\logG	-4	-3	-2	-1	0	1
-1	93.52	55.63	20.06	17.51	11.54	11.54
0	95.37	78.09	37.77	18.58	11.56	11.54
1	95.53	79.53	41.47	19.11	11.59	11.54
2	95.51	79.53	41.47	19.11	11.59	11.54
3	95.51	79.53	41.47	19.11	11.59	11.54

logC\logG	-4	-3	-2	-1	0	1
-1	473.687685s	660.411835s	646.904403s	652.506851s	694.418368s	658.925346s
0	521.857346s	628.632582s	642.169975s	657.648517s	676.84238s	651.366528s
1	507.676706s	615.460444s	631.133505s	633.318898s	675.601806s	643.318241s
2	516.265296s	623.407227s	636.805535s	638.613572s	700.86183s	666.580693s
3	515.311089s	620.961889s	627.407607s	637.963263s	681.222372s	642.294093s

Results on Actual Training and Testing data

Using logC= 1 i.e. C= 2, logG= -4 i.e. g= 0.0625
Accuracy= 95.97%
Training Time= 143.789796s
Testing Time= 68.658342s

On trying a few values near the CV results, the highest accuracy result for C= 2, g=0.03125 is
Accuracy= 97.05%
Training Time= 50.526872s
Testing Time= 50.380531s



Cross Validation Result for Polynomial 2 Kernel

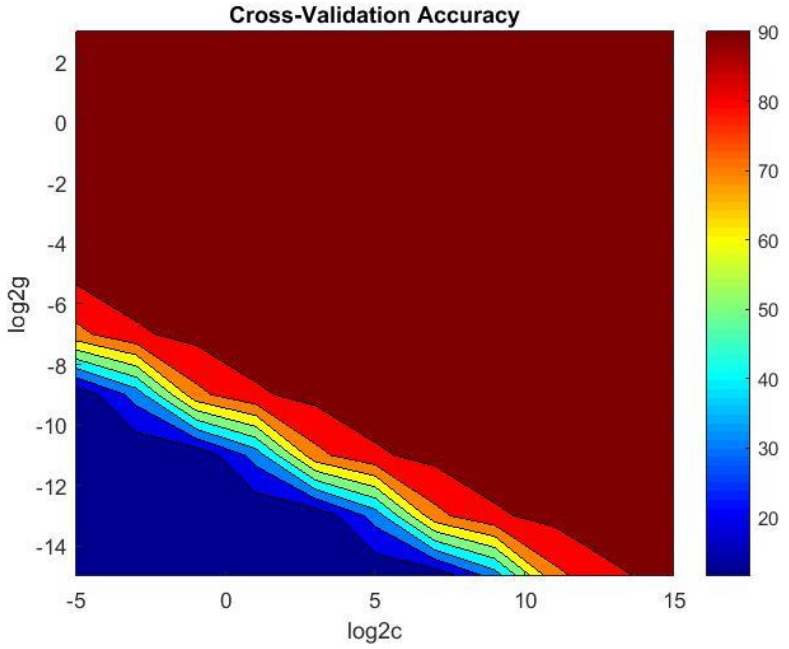
logC\logG	-15	-13	-11	-9	-7	-5	-3	-1	1	3
-5	11.54	11.54	11.54	11.54	77.05	92.92	95.69	95.66	95.66	95.66
-3	11.54	11.54	11.54	34.06	88.68	95.05	95.77	95.66	95.66	95.66
-1	11.54	11.54	11.54	77.05	92.92	95.69	95.66	95.66	95.66	95.66
1	11.54	11.54	34.06	88.68	95.05	95.77	95.66	95.66	95.66	95.66
3	11.54	11.54	77.05	92.91	95.69	95.66	95.66	95.66	95.66	95.66
5	11.54	34.06	88.69	95.05	95.77	95.66	95.66	95.66	95.66	95.66
7	11.54	77.05	92.91	95.69	95.66	95.66	95.66	95.66	95.66	95.66
9	34.06	88.69	95.05	95.77	95.66	95.66	95.66	95.66	95.66	95.66
11	77.04	92.91	95.69	95.66	95.66	95.66	95.66	95.66	95.66	95.66
13	88.69	95.05	95.77	95.66	95.66	95.66	95.66	95.66	95.66	95.66
15	92.91	95.69	95.66	95.66	95.66	95.66	95.66	95.66	95.66	95.66

Results on Actual Training and Testing data

Using logC= -3 i.e. C= 0.125, logG= -3 i.e. g= 0.125
Accuracy= 96.25%
Training Time= 16.363594s
Testing Time= 24.065958s

On changing the coef0 parameter, the highest accuracy result for C= 0.125, g= 0.125, coef0= 0.6 is
Accuracy= 96.35%
Training Time= 15.875508s
Testing Time= 24.732998s

(Another accuracy result:
C= 0.0078125, g= 0.5, coef0= 2.5 is
Accuracy= 96.35%
This result lies above the high cv accuracy value at logC= -3, logG= -3)



Cross Validation Result for Polynomial 4 Kernel

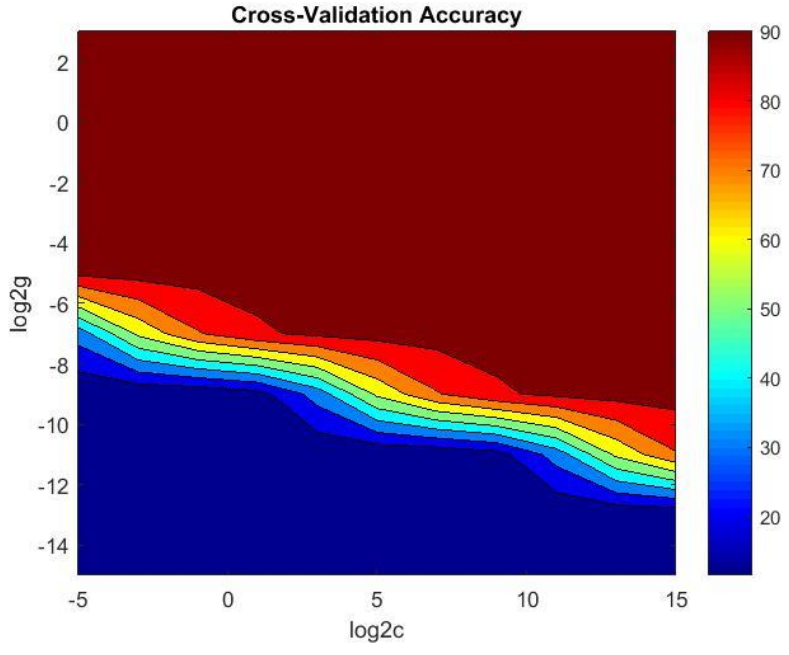
logC\logG	-15	-13	-11	-9	-7	-5	-3	-1	1	3
-5	11.54	11.54	11.54	11.54	34.44	92.29	93.63	93.62	93.62	93.62
-3	11.54	11.54	11.54	11.54	62.09	93.79	93.62	93.62	93.62	93.62
-1	11.54	11.54	11.54	11.73	79.21	93.92	93.62	93.62	93.62	93.62
1	11.54	11.54	11.54	15.87	88.49	93.71	93.62	93.62	93.62	93.62
3	11.54	11.54	11.54	34.44	92.29	93.63	93.62	93.62	93.62	93.62
5	11.54	11.54	11.54	62.09	93.79	93.62	93.62	93.62	93.62	93.62
7	11.54	11.54	11.73	79.21	93.92	93.62	93.62	93.62	93.62	93.62
9	11.54	11.54	15.87	88.49	93.71	93.62	93.62	93.62	93.62	93.62
11	11.54	11.54	34.44	92.29	93.63	93.62	93.62	93.62	93.62	93.62
13	11.54	11.54	62.09	93.79	93.62	93.62	93.62	93.62	93.62	93.62
15	11.54	11.73	79.21	93.92	93.62	93.62	93.62	93.62	93.62	93.62

Results on Actual Training and Testing data

Using logC= -1 i.e. C= 0.5, logG= -5 i.e. g= 0.03125
Accuracy= 94.6%
Training Time= 21.728361s
Testing Time= 23.088042s

On changing the coef0 parameter, the highest accuracy result for C= 0.5, g= 0.03125, coef0= 1.2 is
Accuracy= 96.21%
Training Time= 17.448838s
Testing Time= 24.482072s

(Few more accuracy results:
1) C= 0.05, g= 0.03125, coef0= 1.2 is
Accuracy= 96.21%
2) C= 128, g= 0.0078125, coef0= 0.4 is
Accuracy= 96.15%
This result is near the high cv accuracy value at logC= 7, logG= -7)



Discussion

	RBF	Poly 2	Poly 4	Linear
Accuracy	97.05%	96.35%	96.21%	93.20%
Parameters	C= 2 g= 0.03125	C= 0.125 g= 0.125 coef0= 0.6	C= 0.5 g= 0.03125 coef0= 1.2	C= 0.05
Training Time	50.526872s	15.875508s	17.4488	16.231007s
Testing Time	50.380531s	24.732998s	24.4821	27.544710s
Number of SV	4989	2820	2794	2922
Accuracy on original 60,000 MNIST data	98.56%	98.14%	98.28%	94.71%

1. RBF requires high training and testing time but gives better accuracy than the rest.
2. The training and testing time for Polynomial kernel is comparable to Linear kernel but Polynomial gives better accuracy
3. The training time increases with cost of slack variables.
4. The testing time is mostly dependent on the number of support vectors. More number of SVs more is the testing time.
5. The RBF kernel results in large number of support vectors as compared to other techniques.
6. Since the data has 28 dimensions, it is equivalent to being mapped to a higher dimensional space, therefore the linear kernel also performs considerably well on this data.
7. Cost and Gamma parameter selection play a very important role in finding the model that gives maximum accuracy since according to the chosen value the model may overfit or underfit the data. Thus grid search on possible ranges of C and gamma using cross validation helps us find the values at which the accuracy is maximum and error is lowest.
8. Increasing cost beyond a limit increases error due to variance as the kernel tries to find the model that tries to fit the training data as it is, leading to overfitting. Similarly decreasing cost beyond a limit increases error due to bias as the kernel can move a lot of data to maximize the margin leading to underfitting.
9. If gamma is too large, the radius of the area of influence of the support vectors only includes the support vector itself which leads to overfitting. When gamma is very small, the model does not capture the complexity or shape of the data. The region of influence of any support vector includes the whole training set.
10. Results of polynomial kernel show that increase in error due to increasing the cost can be nullified by decreasing the gamma value since if one makes the model complex the other makes it less complex and vice versa.
11. The coef0 parameter is also very important in polynomial kernel. For certain coef0 values the Accuracy was high for large range of gamma and cost values. I tried polynomial degree 4 kernel for the same range of gamma and cost values and the accuracy was consistently above 94%.

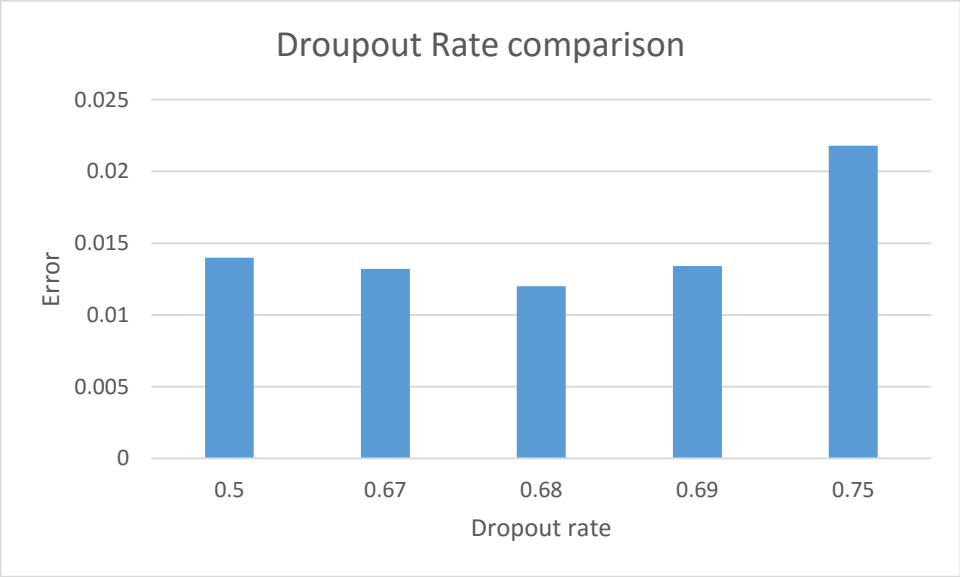
MatConvNet Experiments

Algorithm

1. Download and prepare the mnist data for building the model
2. Use this formula to decide the network components (W – F + 2P)/S + 1. calculateFilters.m used for this purpose
3. Build a pattern that looks something like this
   Input -> [Conv -> Relu -> Pool]*2 -> FC -> Relu -> FC
   Input -> [Conv -> Relu -> Conv -> Relu -> Pool]*2 -> FC -> Relu -> FC
4. Add dropout layer by trying rate values between 0.5 to 1 whichever gives best accuracy

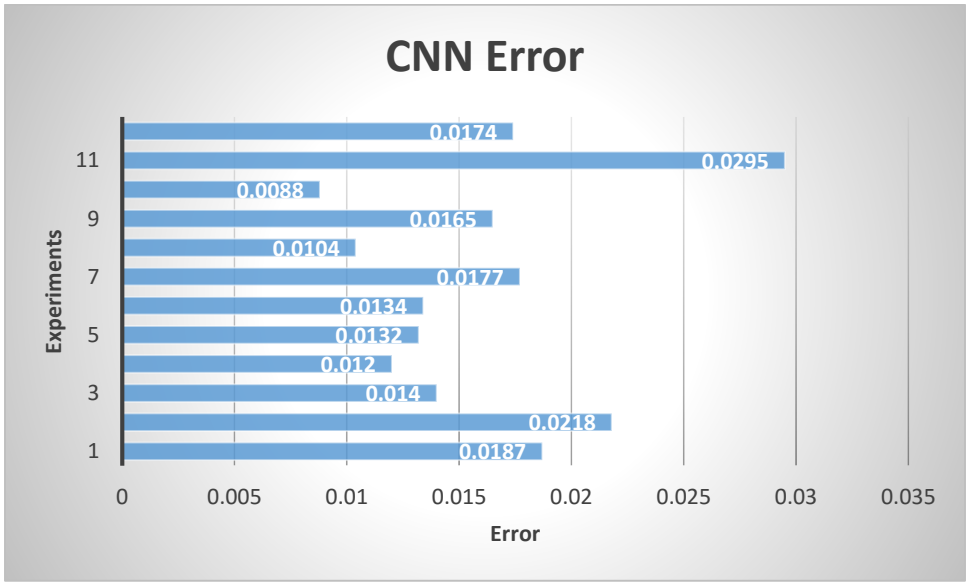
Discussion

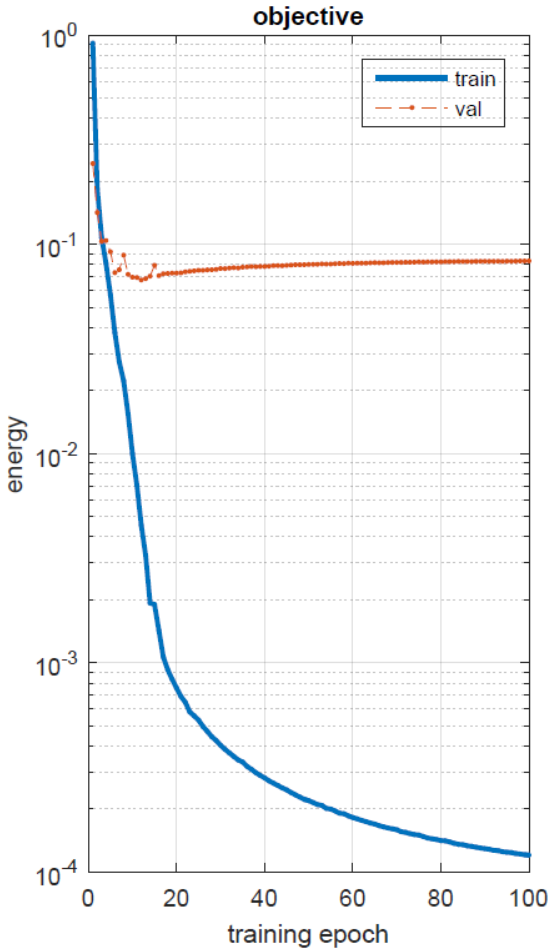
1. The dropout layer reduces overfitting i.e. it reduces error due to variance. This is done by randomly removing certain links and checking the result on validation set.
2. The validation error decreases as a result of adding dropout and training error or the bias increases. But this depends on choosing the correct value of rate as after a point the bias increases.
3. The dropout layer can be configured using a gradient descent approach as for a particular rate the error is minimum and it increases on both the sides.

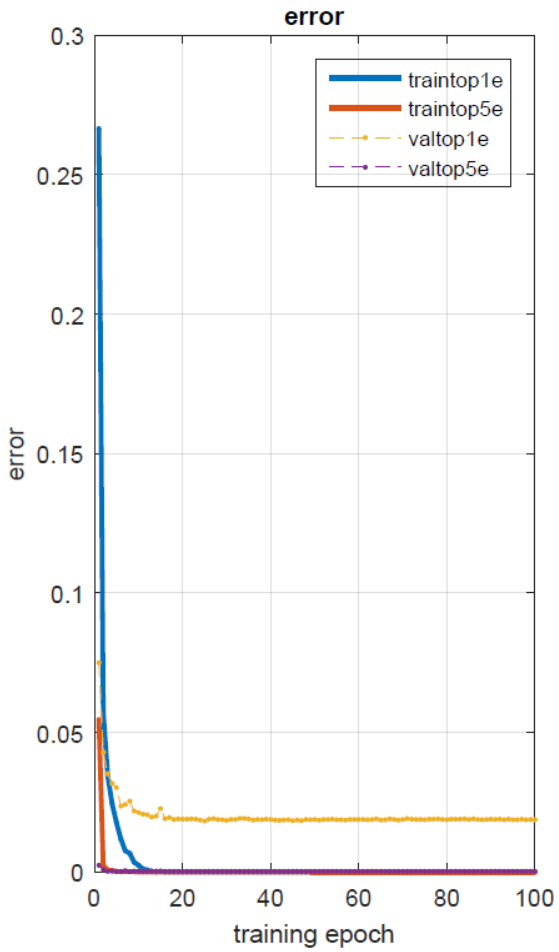


4. For the cases where dropout isn't used the training error becomes zero after very few iterations and the validation error also stabilizes after that. But after adding dropout layer the training error takes much more iterations to reduce to zero. The main focus of the network is to decrease the validation error. With each iteration the validation and training error keep reducing.
5. The CNN is giving better performance with following kind of patterns especially if the Relu layers are followed with dropout
   Input -> [Conv -> Relu -> Pool]*2 -> FC -> Relu -> FC
   Input -> [Conv -> Relu -> Conv -> Relu -> Pool]*2 -> FC -> Relu -> FC
6. As the number of layers or the number of nodes in layers is increased the computation time increases. It increases a bit on the addition of dropout layer as well.
7. Keeping the same network and adding more nodes is improving the performance as in the case of exp12
8. The Pooling layer compresses the input and thus improves execution time and reduces overfitting

Experiments	Changes	Error
Experiment1	No changes	0.018700000
Experiment2	Dropout Layer rate = 0.75	0.021800000
Experiment3	Dropout Layer rate = 0.5	0.014000000
Experiment4	Dropout Layer rate = 0.68	0.012000000
Experiment5	Dropout Layer rate = 0.67	0.013200000
Experiment6	Dropout Layer rate = 0.69	0.013400000
Experiment7	Conv (5*5) -> Relu -> Pool -> Conv (5*5) -> Relu -> Pool -> Conv (4*4) -> Relu -> Conv (1*1)	0.017700000
Experiment8	Conv (5*5) -> Relu -> Dropout -> Pool -> Conv (5*5) -> Relu -> Dropout -> Pool -> Conv (4*4) -> Relu -> Dropout-> Conv (1*1)	0.010400000
Experiment9	Conv (5*5) -> Relu -> Conv (5*5) -> Relu -> Pool -> Conv (3*3) -> Relu -> Pool -> Conv (4*4) -> Relu -> Conv (1*1)	0.016500000
Experiment10	**Conv (5*5) -> Relu -> Dropout -> Conv (5*5) -> Relu -> Dropout -> Pool -> Conv (4*4) -> Relu -> Dropout -> Conv (4*4) -> Relu -> Dropout -> Pool -> Conv (2*2) -> Relu -> Dropout -> Conv (1*1)**	**0.0088000000**
Experiment11	Removed one conv-pool layer	0.029500000
Experiment12	Original Network with more nodes	0.017400000

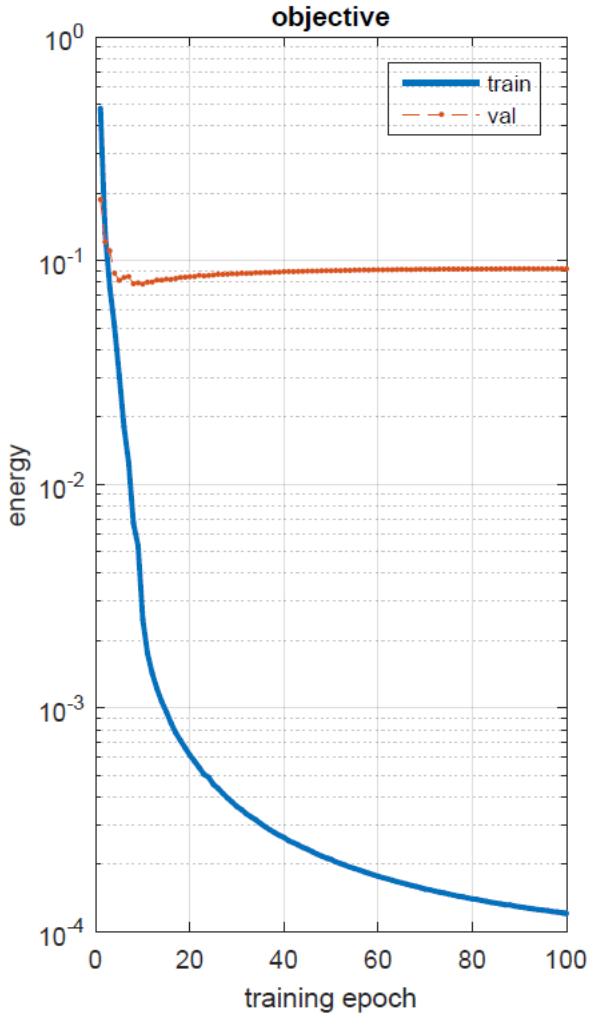


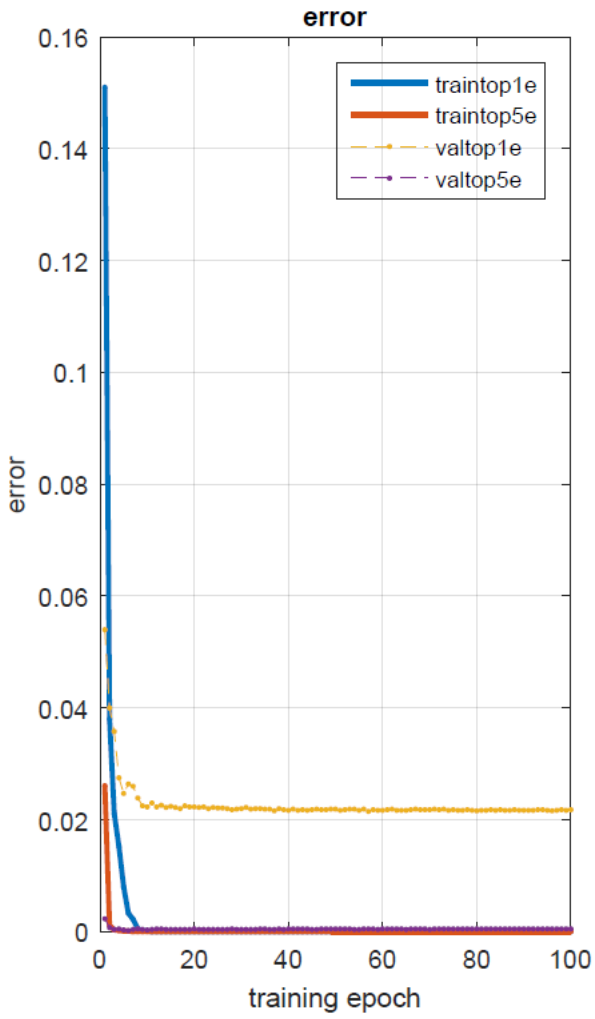




Experiment 1

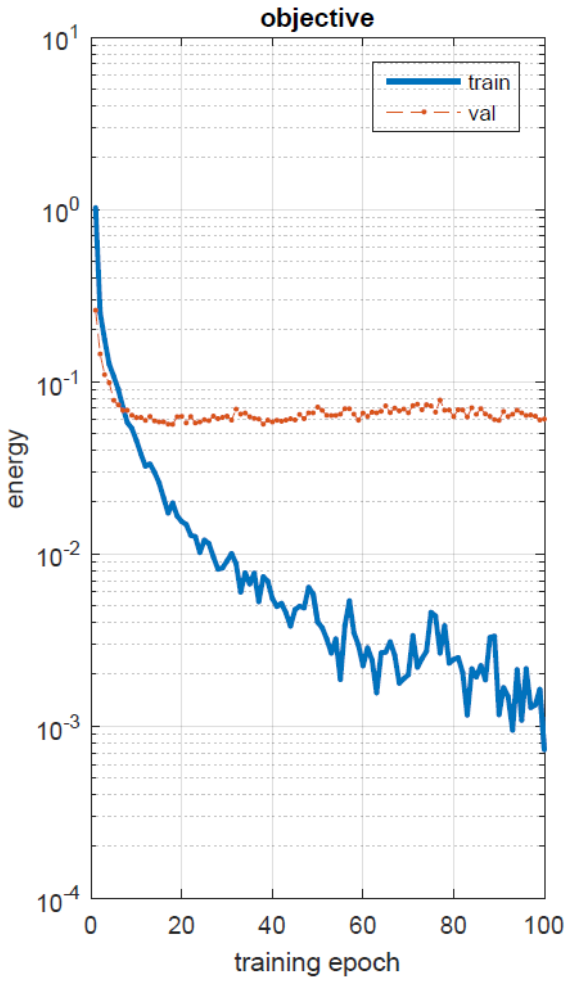
Original Network

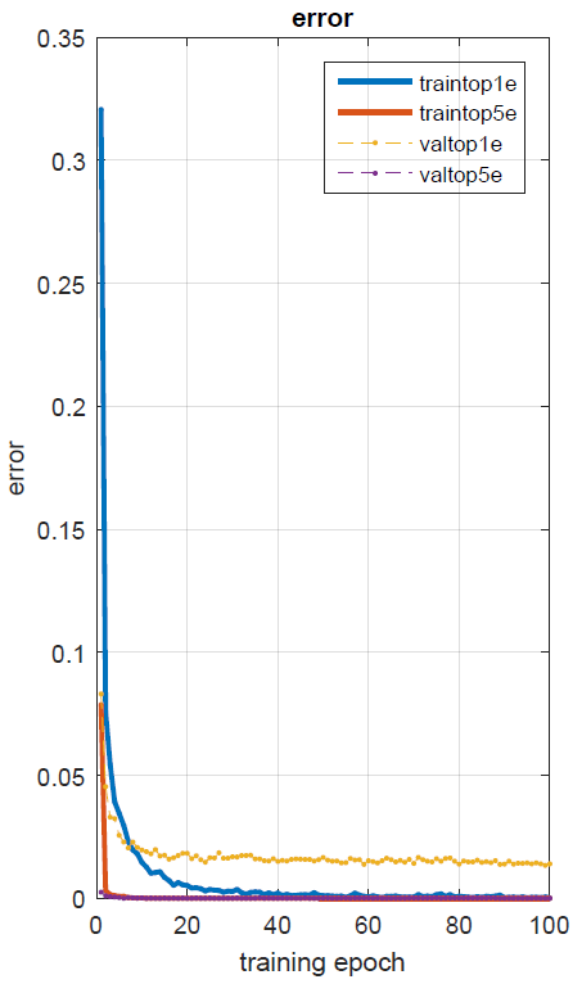




Experiment 2

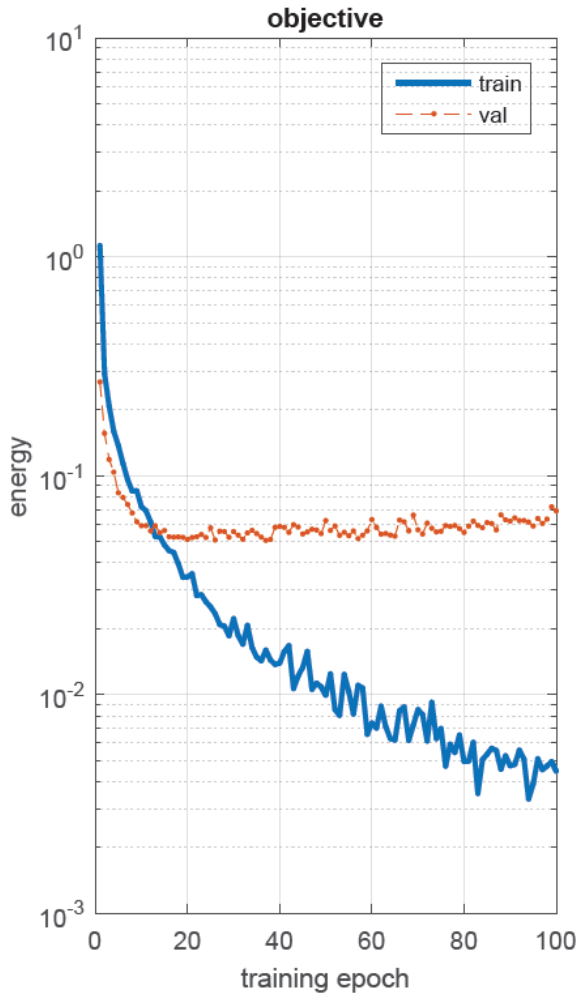
Dropout Layer rate = 0.75

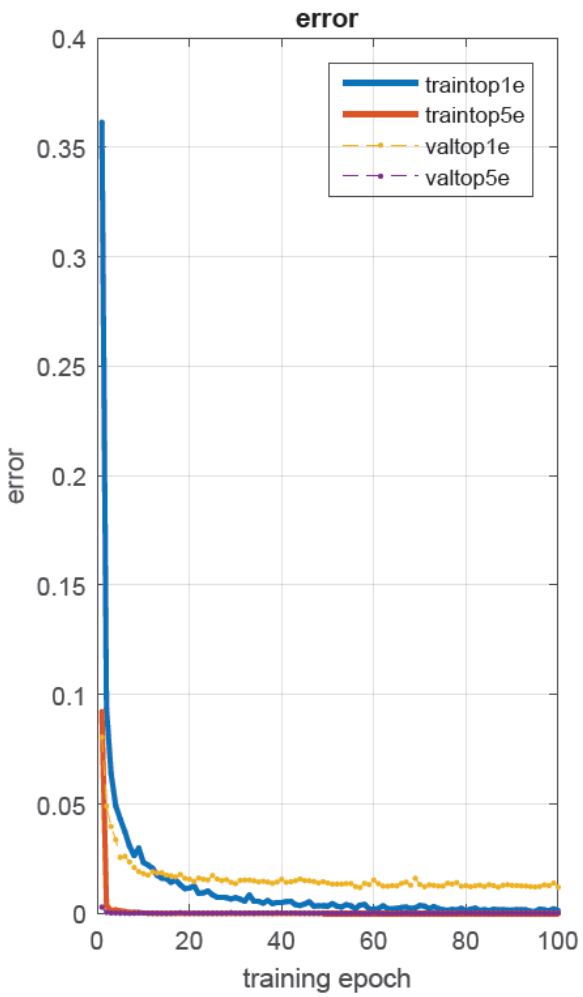




Experiment 3

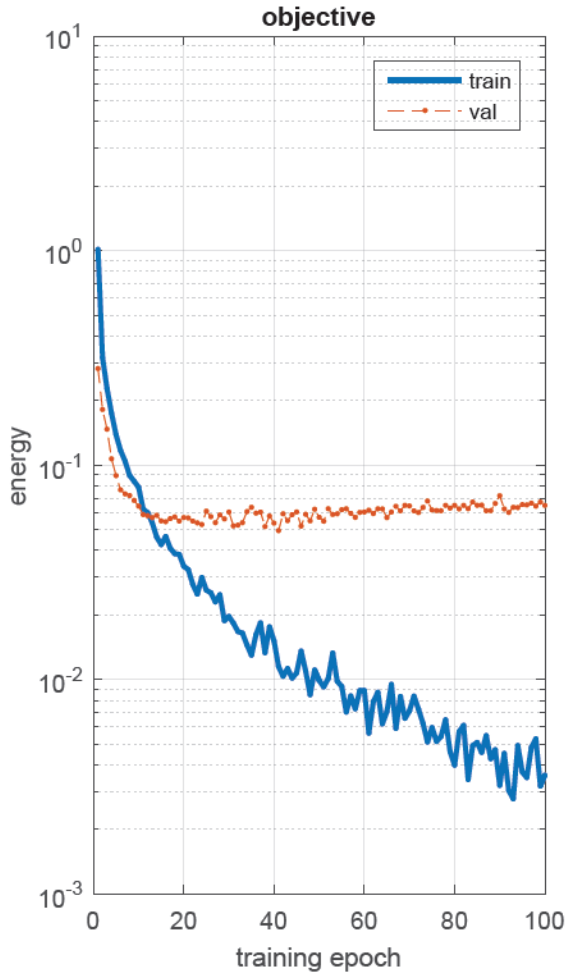
Dropout Layer rate = 0.5

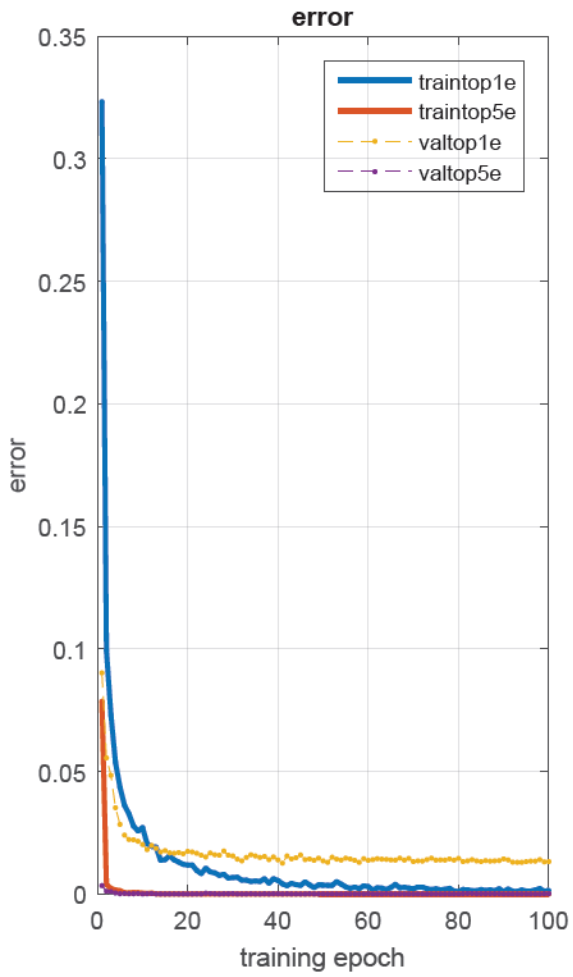




Experiment 4

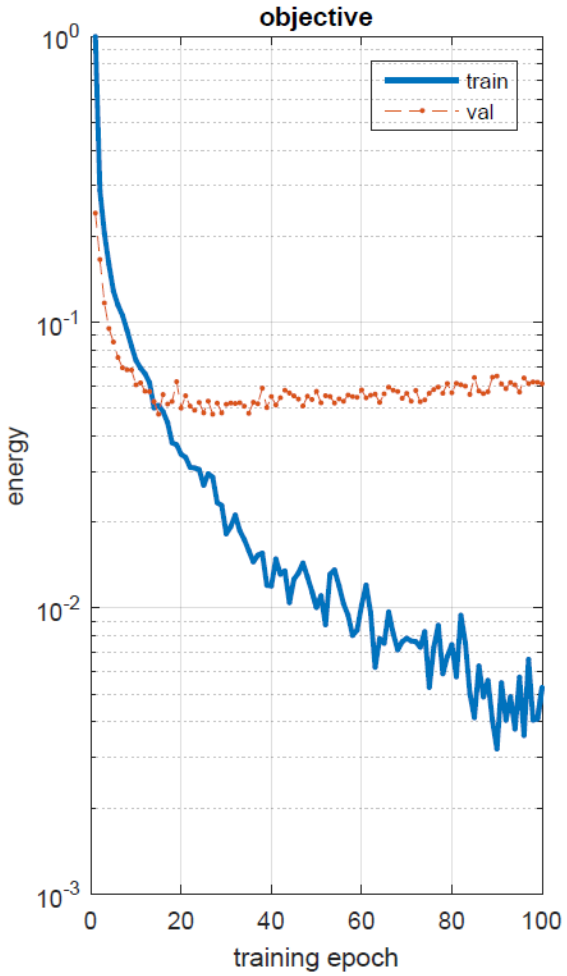
Dropout Layer rate = 0.68

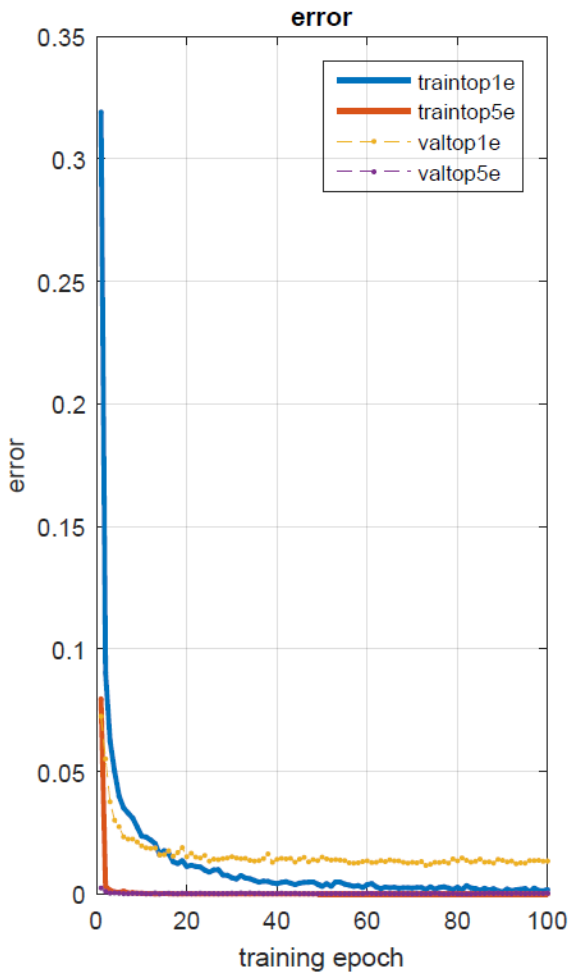




Experiment 5

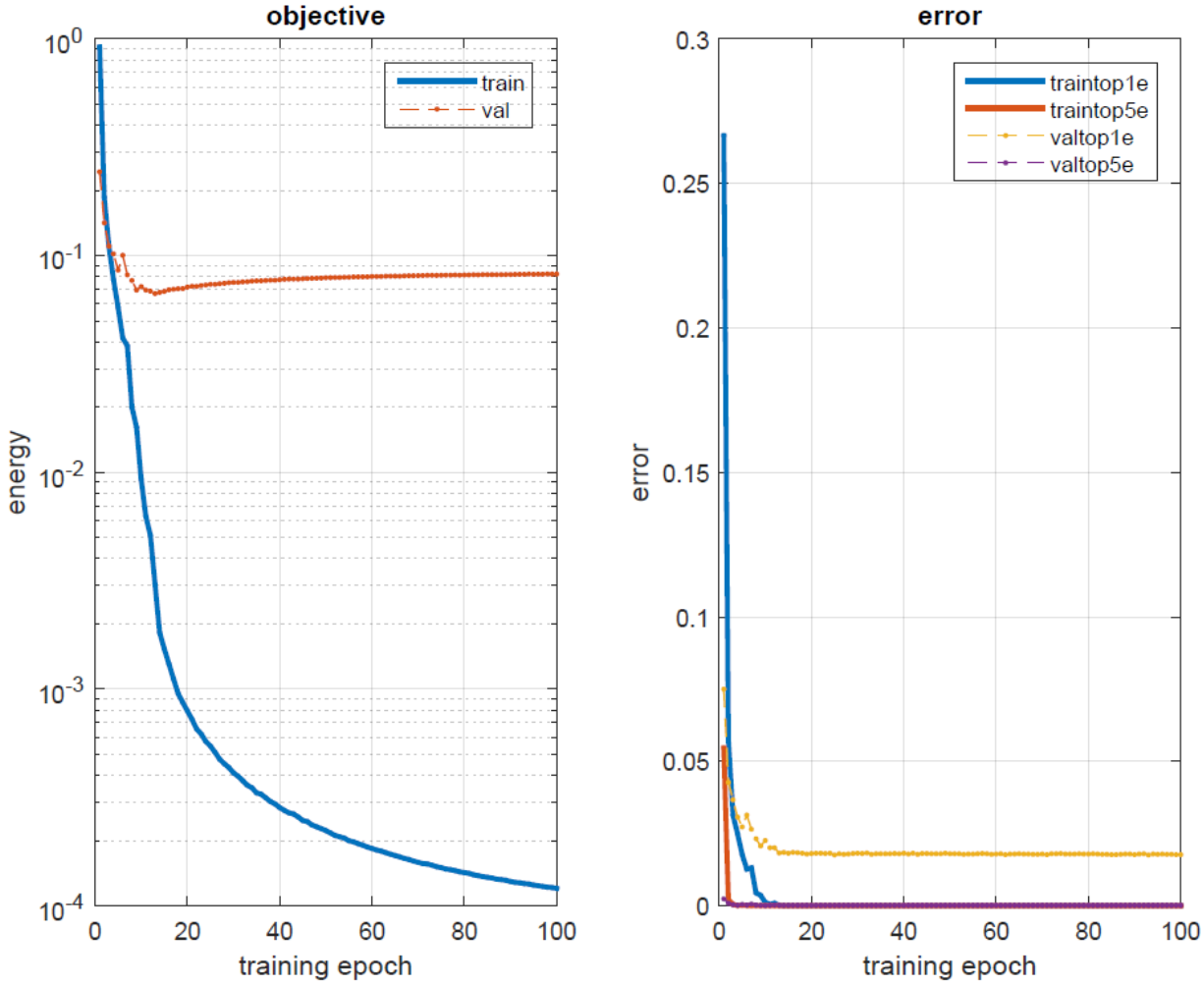
Dropout Layer rate = 0.67





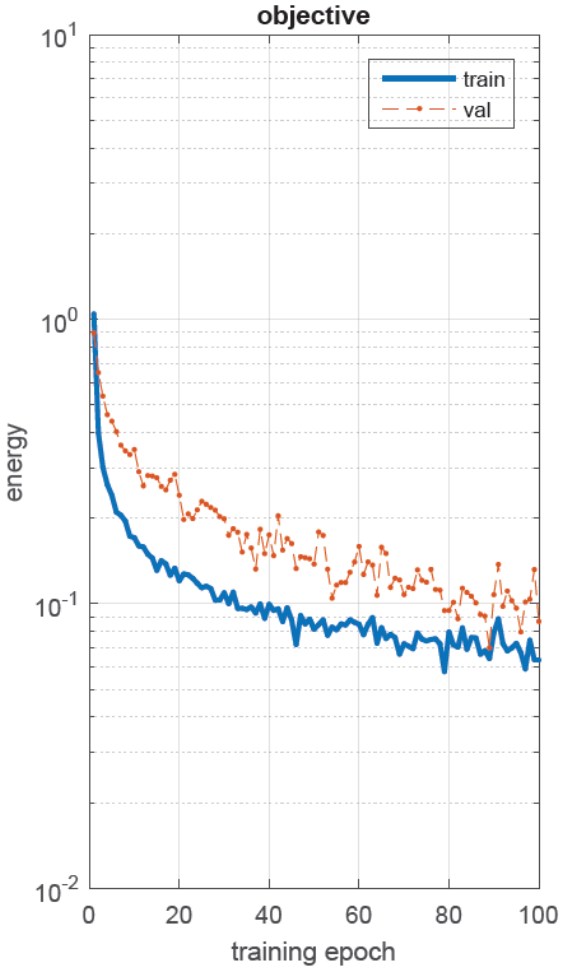
Experiment 6

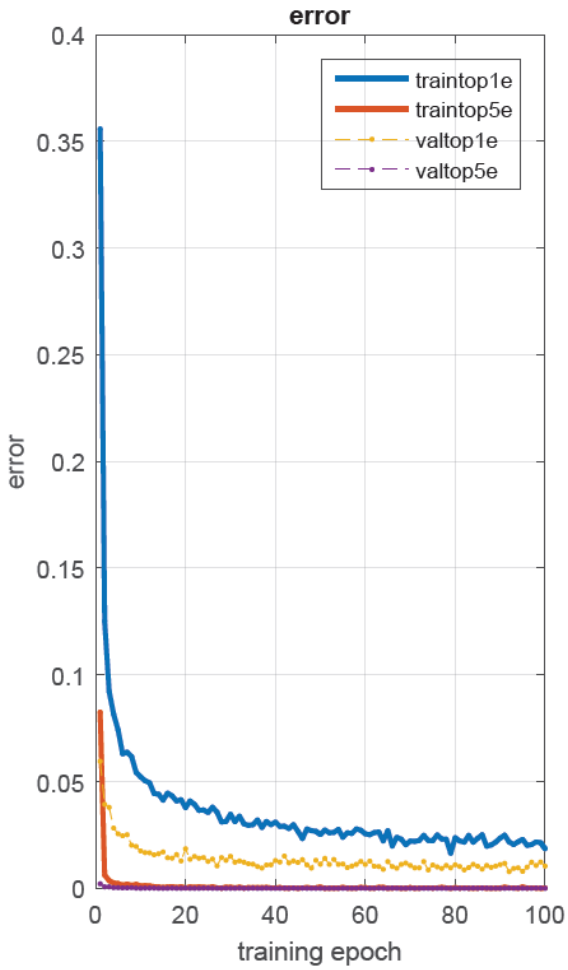
Dropout Layer rate = 0.69



Experiment 7

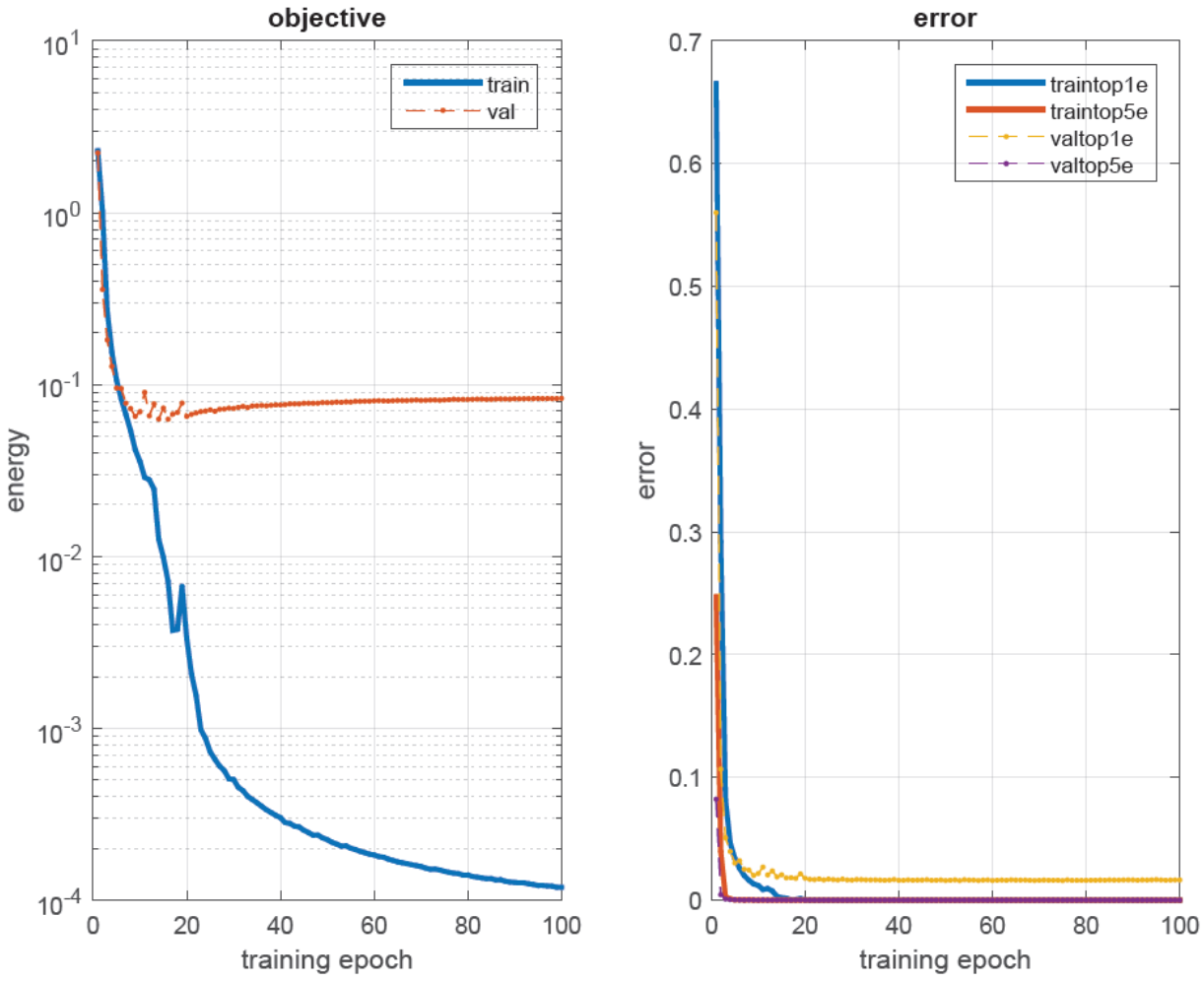
Input -> [Conv -> Relu -> Pool]*2 -> FC -> Relu -> FC





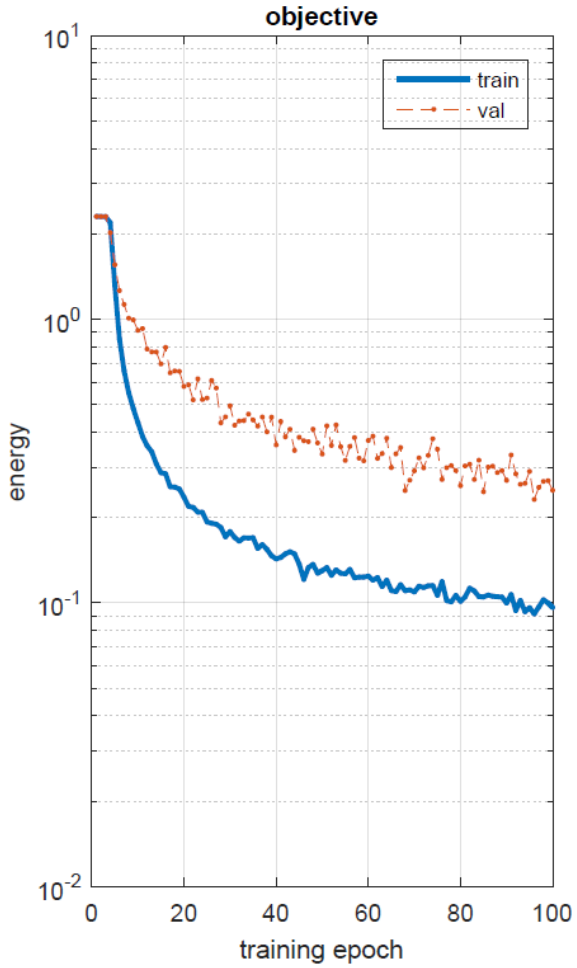
Experiment 8

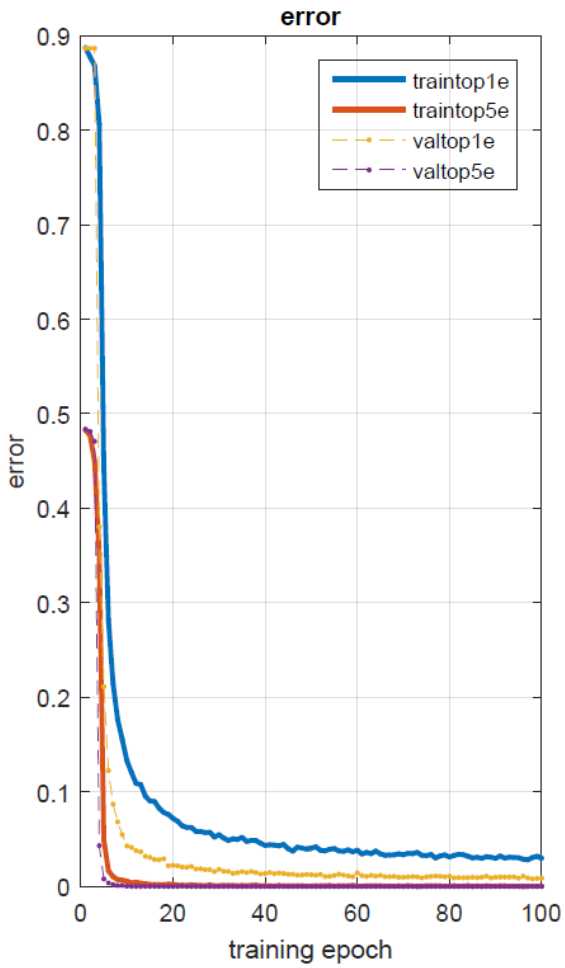
Input -> [Conv -> Relu -> Pool]*2 -> FC -> Relu -> FC with dropout



Experiment 9

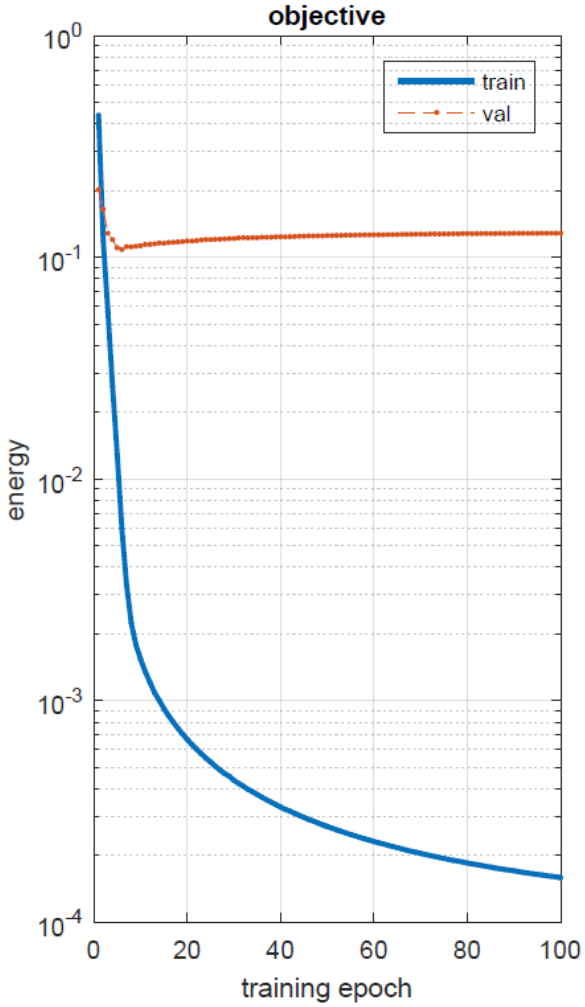
Input -> [Conv -> Relu -> Conv -> Relu -> Pool]*2 -> FC -> Relu -> FC

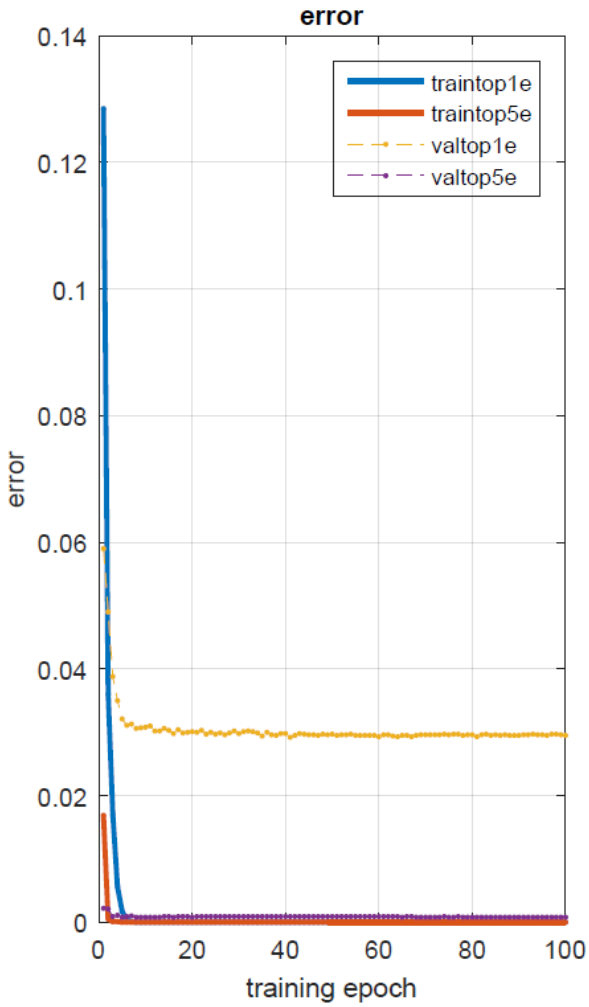




Experiment 10

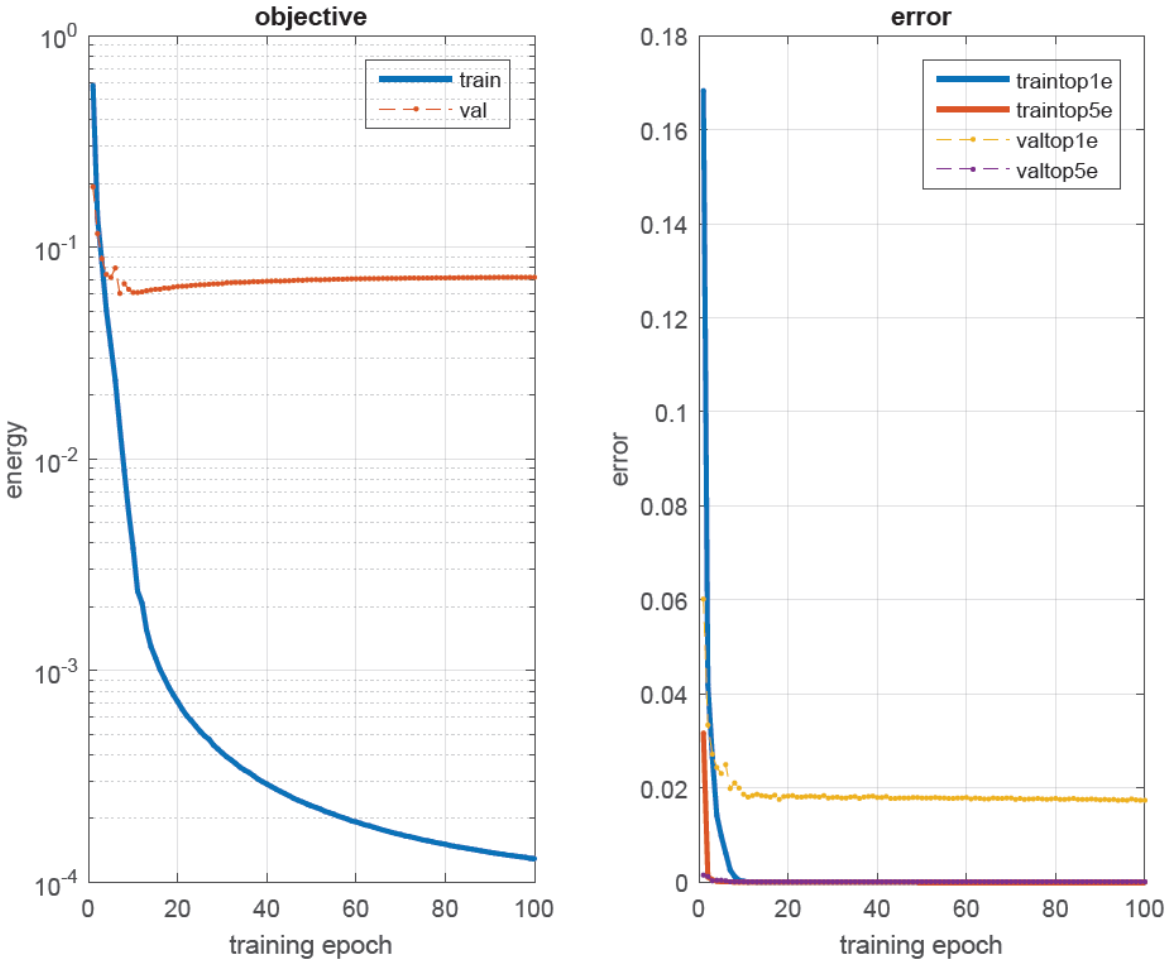
Input -> [Conv -> Relu -> Conv -> Relu -> Pool]*2 -> FC -> Relu -> FC with dropout





Experiment 11

Removed one conv-pool layer



Experiment 12

Original Network with more nodes