Mini Project 4: Convolutional Neural Networks

Siddhant Prakash
1211092724
sprakas9@asu.edu

***Abstract*—Report for mini-project 4 to train a CNN architecture on SVHN dataset, to get optimal performance, with more speed and less memory usage.**

1. Introduction

In this mini-project we are provided with the YANN [1] toolbox using which, we train a convolutional neural network on the SVHN [2] data set. The aim of the project is to come up with an architecture and optimizations which will give us the best accuracy on the dataset. Some constraints are imposed such that the network should optimize the memory usage as well as, testing time should be low.

2. Dataset and Training

The input to the network is the SVHN data set which has images of size 32X32 with 3 channels. There are 630,420 images in the data set, of which we discard 420. The remaining data set was shuffled and divided in batches of 500. The summary of data set can be seen in Table 1. Each batch was divided into mini-batches of 10 data points. Hence, each training, validation and testing had 56, 28 and 42 mini-batches.

Total Batches	630,000 / 500 = 1,260
Test Batches	420
Train Batches	560
Valid Batches	280

TABLE 1. DATA SET DISTRIBUTION

The training was done on an Ubuntu 14.04 system with 1.7 GHz Intel Core i5 processor, 8GB 1600 MHz DDR3L SD-RAM and hosting Nvidia GeForce 840M GPU. All training and testing, memory and time is reported on GPU.

3. Experiments on Network Architectures

The networks that were experimented are listed in Table 7, 8, 9, 10, 11, 12, 13 and 14. Each architecture leads to the next architecture with slight modification. Initially, we started with the best network and parameters that we got for mini-project 3, and used the same architecture (Table 7) on this new data set. The idea for the next architecture in Table 8 was taken from LeNet [7] and layers were subsequently added to test for optimum architecture. We have used batch normalization [8] on all convolutional and conv-pool layers. The activation function for all convolutional, conv-pool and fully connected layers is 'ReLU'. The initial parameters settings were as listed in Table 2.

Epoch	Net 1-2 :(15,15) Net 3-8 :(10,10)
Momentum	'Nesterov'
Momentum Rate	(0.6,0.95, 30)
Optimizer	'Rmsprop'
Regularization Const.	(0.0001,0.0002)
Learning Rate	(0.01,0.001,0.0001)

TABLE 2. INITIAL PARAMETERS

For network 1 and 2, the parameters were kept the same. Although, in training of 1 and 2, we observed the data set starts over-fitting at around 15 epochs. So we reduced down the epochs to 20 with 2 eras of 10 epochs each with momentum rate changed to (0.6,0.95, 20). Thus, post network 1 and 2, all architectures were tested with same settings. We increased number of layers until we got memory in excess limitation. With the remaining network that we could work, we changed the number of neurons in different layers and monitored the results.

3.1. Results

Keeping the parameters as described in Table 2, the results for accuracy, and testing time (best reported) is summarized in Table 3

Analysis. We selected **network 3 (Table 9)** as the optimum network based on the highest accuracy along with optimal testing time as well as memory usage amongst all the networks. Although, network 7 and 8, which has the same architecture, but different number of neurons were also good candidates, still we go with network 3.

4. Experiments on Optimizers

Now, we fixed the architecture of network 3, and experimented with different epochs, momentum and optimizer setting. Keeping the momentum and optimizer same as Table 2, we change the epochs to 10, with 2 era of 5 epochs each as the best validation accuracy we were getting was still on epoch 10, after which the network started over-fitting. We also tested with a single era of 10 epochs with learning rate of (0.01,0.001).

Network	Accuracy (%)	Testing Time (mins.)	Memory Usage (MiB)
Net 1 [Tab7]	91.48	**1:18**	~250
Net 2 [Tab8]	97.24	2:47	~300
Net 3 [Tab9]	**97.91**	2:34	~600
Net 4 [Tab10]	97.44	3:58	~1000
Net 5 [Tab11]	-	-	~1600* (Exceed Mem)
Net 6 [Tab12]	97.25	1:27	~1000
Net 7 [Tab13]	97.28	2:56	~600
Net 8 [Tab14]	97.60	2:51	~600

TABLE 3. RESULTS FOR EXPERIMENT WITH DIFFERENT ARCHITECTURES

For momentum and optimizers, we tested with Polyak [3] and Nesterov [4] momentum, with Adagrad [6] and Rmsprop [5] optimizers.

4.1. Results

The results for experiment on epochs is listed in Table 4 and with different momentum and optimizers are listed in 5.

Epochs	Accuracy(%)	Testing Time (mins.)
(10,10)	97.59	**2:51**
(5,5)	**97.60**	2:53
(10)	97.22	2:53

TABLE 4. EXPERIMENTS WITH NUMBER OF EPOCHS

Momentum	Optimizer	Accuracy(%)	Testing Time (mins.)
Polyak	Adagrad	91.25	2:36
Polyak	Rmsprop	97.34	2:49
Nesterov	Adagrad	93.38	**2:33**
Nesterov	Rmsprop	**97.54**	2:36

TABLE 5. EXPERIMENTS WITH OPTIMIZERS

Analysis. As accuracy does not get affected with decreasing the epochs, but the reduce in training time is significant, we can convincingly say, running for 10 epochs with 2 era of 5 epochs each **(5,5)** with a learning rate of **(0.01, 0.001, 0.0001)**, is a good choice. We also increased the starting rate for momentum and finally we achieved best results for momentum rate of **(0.7, 0.95, 30)**.

With regard to optimizers, it is clear that '**Rmsprop**' fare way better than 'Adagrad'. For momentum, we can choose either of 'Polyak' or 'Nesterov', but '**Nesterov**' does seem to have an edge over 'Polyak' momentum.

5. Discussions

Summing it all together, we find the network that performs the best on the given SVHN full data set is described in Table 9. Thus, we use 3 conv-pool layers of filter size 5X5, 3X3 and 3X3 and number of feature maps being 60, 150 and 270 respectively. The output of these layers are flattened out and fed to 2 fully connected layers, each having 800 hidden layers, which is connected to a 'Softmax' classifier, that gives a probability of the image being in one of the 10 classes. The best set of parameters, which optimizes the chosen architecture to keep it well under the constraints imposed, is given in Table 6.

Epoch	(5,5)
Momentum	'Nesterov'
Momentum Rate	(0.7, 0.95, 30)
Optimizer	'Rmsprop'
Regularization Const.	(0.0001, 0.0002)
Learning Rate	(0.01, 0.001, 0.0001)

TABLE 6. FINAL PARAMETERS

There are still a number of parameters and architectures that are left to explore. We have essentially only explored the basic LeNet architecture. We would have liked to explore AlexNet inspired architecture with 'Dropout' layers, and subsequently with more deeper networks such as 'GoogleNet' and 'Residual Networks'.

References

[1] Yet Another Neural Network [YANN] Toolbox. https://github.com/ragavvenkatesan/yann

[2] Netzer, Yuval, et al. "Reading digits in natural images with unsupervised feature learning." NIPS workshop on deep learning and unsupervised feature learning. Vol. 2011. No. 2. 2011.

[3] Polyak, Boris T. "Some methods of speeding up the convergence of iteration methods." USSR Computational Mathematics and Mathematical Physics 4.5 (1964): 1-17.

[4] Nesterov, Yurii. "A method of solving a convex programming problem with convergence rate O (1/k2)." Soviet Mathematics Doklady. Vol. 27. No. 2. 1983.

[5] Tieleman, Tijmen, and Geoffrey Hinton. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude." COURSERA: Neural networks for machine learning 4.2 (2012).

[6] Duchi, John, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." Journal of Machine Learning Research 12.Jul (2011): 2121-2159.

[7] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.

[8] Ioffe, Sergey, and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167 (2015).

Layer No.	Input Shape	Receptive Field	No. of Feature Maps	Type of Neurons
1	32X32X3 = 3072	1X1	800	Fully Connected
2	800	1X1	800	Fully Connected
3	800	1X1	10	Softmax

TABLE 7. NETWORK ARCHITECTURE I

Layer No.	Input Shape	Receptive Field	No. of Feature Maps	Type of Neurons
1	32X32X3	5X5	20	Convolutional
	28X28X20	2X2		Pool
2	14X14X20	5X5	40	Convolutional
	10X10X40	2X2		Pool
3	5X5X40 = 1000	1X1	500	Fully Connected
4	500	1X1	10	Softmax

TABLE 8. NETWORK ARCHITECTURE II

Layer No.	Input Shape	Receptive Field	No. of Feature Maps	Type of Neurons
1	32X32X3	5X5	60	Convolutional
	28X28X60	2X2		Pool
2	14X14X60	3X3	150	Convolutional
	12X12X150	2X2		Pool
3	6X6X150	3X3	270	Convolutional
	4X4X270	2X2		Pool
4	2X2X270 = 1080	1X1	800	Fully Connected
5	800	1X1	800	Fully Connected
6	800	1X1	10	Softmax

TABLE 9. NETWORK ARCHITECTURE III

Layer No.	Input Shape	Receptive Field	No. of Feature Maps	Type of Neurons
1	32X32X3	5X5	60	Convolutional
	28X28X60	2X2		Pool
2	14X14X60	3X3	150	Convolutional
	12X12X150	2X2		Pool
3	6X6X150	3X3	270	Convolutional
	4X4X270	1X1		
4	4X4X270	3X3	360	Convolutional
	2X2X360	1X1		
5	2X2X360 = 1440	1X1	800	Fully Connected
6	800	1X1	800	Fully Connected
7	800	1X1	10	Softmax

TABLE 10. NETWORK ARCHITECTURE IV

Layer No.	Input Shape	Receptive Field	No. of Feature Maps	Type of Neurons
1	32X32X3	5X5	20	Convolutional
	28X28X20	2X2		Pool
2	14X14X20	3X3	40	Convolutional
	12X12X40	1X1		
3	12X12X40	3X3	60	Convolutional
	10X10X60	1X1		
4	10X10X60	3X3	150	Convolutional
	8X8X150	1X1		
5	8X8X150 = 9600	1X1	4800	Fully Connected
6	4800	1X1	1000	Fully Connected
7	1000	1X1	10	Softmax

TABLE 11. NETWORK ARCHITECTURE V

Layer No.	Input Shape	Receptive Field	No. of Feature Maps	Type of Neurons
1	32X32X3	5X5	20	Convolutional
	28X28X20	2X2		Pool
2	14X14X20	3X3	40	Convolutional
	12X12X40	2X2		Pool
3	6X6X40	3X3	60	Convolutional
	4X4X60	1X1		
4	4X4X60	3X3	150	Convolutional
	2X2X150	1X1		
5	2X2X150 = 600	1X1	600	Fully Connected
6	600	1X1	600	Fully Connected
7	600	1X1	10	Softmax

TABLE 12. NETWORK ARCHITECTURE VI

Layer No.	Input Shape	Receptive Field	No. of Feature Maps	Type of Neurons
1	32X32X3	5X5	20	Convolutional
	28X28X20	2X2		Pool
2	14X14X20	3X3	40	Convolutional
	12X12X40	2X2		Pool
3	6X6X40	3X3	150	Convolutional
	4X4X150	2X2		Pool
4	2X2X150 = 600	1X1	600	Fully Connected
5	600	1X1	600	Fully Connected
6	600	1X1	10	Softmax

TABLE 13. NETWORK ARCHITECTURE VII

Layer No.	Input Shape	Receptive Field	No. of Feature Maps	Type of Neurons
1	32X32X3	5X5	60	Convolutional
	28X28X60	2X2		Pool
2	14X14X60	3X3	150	Convolutional
	12X12X150	2X2		Pool
3	6X6X150	3X3	270	Convolutional
	4X4X270	2X2		Pool
4	2X2X270 = 1080	1X1	1080	Fully Connected
5	1080	1X1	800	Fully Connected
6	800	1X1	10	Softmax

TABLE 14. NETWORK ARCHITECTURE VIII