Model Complexity-Accuracy Trade-off for a Convolutional Neural Network

Atul Dhingra

Abstract

Convolutional Neural Networks(CNN) has had a great success in the recent past, because of the advent of faster GPUs and memory access. CNNs are really powerful as they learn the features from data in layers such that they exhibit the structure of the V-1 features of the human brain. A huge bottleneck in this case is that CNNs are very large and have a very high memory footprint, and hence they cannot be employed on devices with limited storage such as mobile phone, IoT etc. In this work we study the model complexity versus accuracy trade-off on MNSIT dataset, and give a concrete framework of handling such a problem, given the worst case accuracy that a system can tolerate.

Index Terms

CNN, MNIST, Speed-accuracy Tradeoff, Tensorflow

Experimental Setup

The experiment on speed-accuracy trade-off for digit recognition using MNIST dataset [1] is done using tensorflow [2]. The data is split into training set that contains $55,000$ data points, testing set containing $10,000$ data points and validation set containing $5,000$ data points. Each image in the dataset is a grayscale image of size $28X28$. The labels are "one-hot vectors", which a very sparse vector that contains 0 in all dimensions except one dimension where it is 1 corresponding to the true identity. In our work we use the sofmax regression model to compute the loss between the predicted and the ground truth class. The model is trained using Adam optimizer with learning rate $1e-4$, and mini-batch size of 50 for a total of 20,000 iterations. These parameters are fixed, as we are studying the trade-off of the model complexity and testing accuracy, and these two parameters only help with the convergence time, which we are not considering in our experiment. The convolutional layers weights and biases are randomly initialized with mean zero($\mu = 0$) and standard deviation $0.1(\sigma = 0.1)$. All the experiments are performed using tensorflow[2], using a NVIDIA GeForce GTX Titan X.

I. Baseline Model

The baseline model consists of two convolutional layers(conv1,conv2) and two fully connected layers(fc1,fc2) with max pooling layer(pool1,pool2) after conv1 and conv2. There is a dropout layer betwen fc1 and fc2 as shown in the table I

TABLE I
Baseline Network

Name	Type	Filter	Output Size	Memory	#Params
Input	Image		28x28x1	28*28*1 =0.784K	0
conv1	Convolution	5x5x1	28x28x32	28*28*32 =25K	(5*5*1)*32 =800
pool1	Max Pooling	2x2	14x14x32	14*14*32 =6.272K	0
conv2	Convolution	5x5x32	14x14x64	14*14*64 =12.5K	(5*5*32)*64 =51,200
pool2	Max Pooling	2x2	7x7x64	7*7*64 =3.1K	0
fc1	Fully Connected		1024	1024	(7*7*64)*1024 =3,211,264
fc2	Fully Connected		10	10	1024*10 =10,240
total				48.68K	3.27M

The memory footprint of this network is approximately 50K and the number of parameters is 3.27M. The testing accuracy achieved on the baseline model is 99.29%, and a run time of 165 seconds on Titan X. The As we know the computation speed of convolutional layer is slow, but its memory footprint is small, as evident from table I, on the other hand, the fully connected layers have a high memory footprint, but computationally, it is faster. Keeping these facts in mind, we improved the existing network in the following section.

II. Optimized Network

Design Framework

As in table I, the highest memory contribution is by conv1 and the highest parameter contribution is by fc1. This suggests that to reduce the memory footprint of the network, we need to tweak the conv layers. As the worst-case baseline test accuracy in our case is 95%, we removed conv2, to see if we can get away with just one convolutional layer, as seen in table II In

TABLE II
Optimization Iteration 1

Name	Type	Filter	Output Size	Memory	#Params
Input	Image		28x28x1	28*28*1 =0.784K	0
conv1	Convolution	5x5x1	28x28x32	28*28*32 =25K	(5*5*1)*32 =800
pool1	Max Pooling	2x2	14x14x32	14*14*32 =6.272K	0
fc1	Fully Connected		1024	1024	(14*14*32)*1024 =6,422,528
fc2	Fully Connected		10	10	1024*10 =10,240

doing so, we managed to decrease the memory footprint by 15K, but the parameters doubled for fc1 to approximately 6.4M. This network gave a test accuracy of 99.27% and ran for 164 seconds on Titan X. To reduce the number of parameters, we varied the output size of the fc1 layer as shown in table III

TABLE III
Varying filter size of fully connected layer

fc1(size)	Test Accuracy	Time(Titan X)
1024	99.27	164 s
512	99.25	155 s
256	99.21	148 s
128	99.1	145 s
64	98.9	159 s
32	98.7	158 s

We observe when fc1 is 128 in size, it gives the best trade-off for running time without affecting the accuracy too much. By choosing 128 as fc1 size, we have dramatically decreased the number of parameters in fc1 from 6.4M in table II to 0.8M in this case. This motivated us to use 128 size fc1 in our final model.

After all these tweaks, still the memory footprint is high, and the number of parameters in millions. To further reduce the size and parameters, we tweaked the conv1 filter size, and varied it to 3x3 and 1x1. we observed that 3x3 gave us the best tradeoff betwen the number of parameters and the testing accuracy. We also varied the depth of the convolutional filter to see if actually need 32 filters to achieve 95% accuracy. The results below use the fc1 layer of size 128, while varying the conv1 layer in table IV,V,VI

TABLE IV
Varying depth on 5x5 conv1 filter

Filter Size	Test Accuracy	Time in seconds(on Titan X)
5x5x32	98.6	114
5x5x16	98.5	110
5x5x8	98.2	107
5x5x4	97.8	109
5x5x2	97.38	108

From this data we observe that the accuracy dips more rapidly as the conv1 filter size is decreased. For example for 5x5 filter in table IV the accuracy dips from 98.6 to 97.38, that is about 1% decrease whereas in 3x3 filter, the accuract dips from 98 to 96.2, which is almost double. Hence for our design, we stick with the 5x5 filter, as in our final design we use a depth 2 filter, and only 32 parameters are added(50-18), which is minuscule as compared to the order of parameters in thousands. We also use a 4x4 pooling layer instead of the 2x2 pooling layer in baseline to reduce the parameters from fc1 layer without losing much test accuracy.

TABLE V
VARYING DEPTH ON 3X3 CONV1 FILTER

Filter Size	Test Accuracy	Time in seconds(on Titan X)
3x3x32	98	117
3x3x16	97.79	108
3x3x8	97.5	103
3x3x4	96.6	108
3x3x2	96.2	106

TABLE VI
VARYING DEPTH ON 1X1 CONV1 FILTER

Filter Size	Test Accuracy	Time in seconds(on Titan X)
1x1x32	95.88	111
1x1x16	95.9	106
1x1x8	95.0	105
1x1x4	94.38	102
1x1x2	94.28	100

TABLE VII
OPTIMIZED NETWORK

Name	Type	Filter	Output Size	Memory	#Params
Input	Image		28x28x1	28*28*1 =0.784K	0
conv1	Convolution	5x5x2	28x28x2	28*28*2 =1.5K	(5*5*1)*2 =50
pool1	Max Pooling	4x4	7x7x2	7*7*4 =98	0
fc1	Fully Connected		128	128	(7*7*2)*128 =12.5K
fc2	Fully Connected		10	10	128*10 =1.2K
total				2.5K	13.8K

Final Model

We designed the following network as shown in table VII as per the worst case threshold of 95% test accuracy. This optimized model produces a test accuracy of 95.81%. This optimized network has 19.5 times smaller memory footprint as compared to the baseline model. It also requires 236 times less parameter computations as compared to the baseline model as detailed in the table VIII

TABLE VIII
COMPARISON OF BASELINE NETWORK WITH OPTIMIZED NETWORK

	Baseline	Optimized
#Params	3.27M	13.8K
Memory	48.65K	2.5K
Run Time(Titan X)	165sec	98sec

Note: With the same architecture, but using 3x3 conv1 instead of 5x5, although we get a reduction of 32 parameters(5*5*2-3*3*2), but the test accuracy goes to 93.66% which violates our design parameters.

III. MODEL SIZE AND ACCURACY COMPARISON

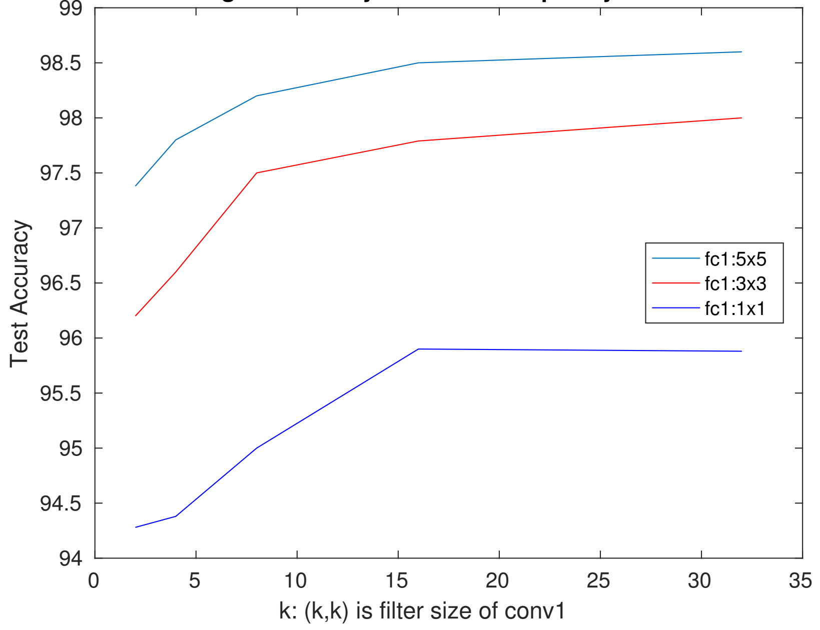
Given $s(f)$ is the minimum model size that can achieve the accuracy f, we can assert that

$$0 < s(f_1) < s(f_2) ... < s(f_n) < 1 \tag{1}$$

such that, $0 \leq f_1 < f_2 < ... < f_n \leq 1$ This is evident from fig ,as the model size decreases, the testing accuracy decreases as shown in fig III

Take a slice of fig III at k=32, at this point the 5x5 curve is above 3x3 , which is above 1x1 curve. at this slice k=32, the complexity of the 5x5 network is greater than 3x3, which is greater than 1x1. Similarly the test accuracy at this point follows the same trend. What we can assert is that, as the other parameters of the model is same, and we decrease the filter size from 5x5 to 3x3 followed by 1x1, the model complexity of the network is decreasing. But, as we see from the fig III, the test accuracy also follows the same trend, therefore we can assert that the claim made in equation 1 is valid. This can also be realized by the fact that the Optimal network proposed in this scenario has a 5x5 filter, which has a test accuracy of 95.81%,

Fig 1: Accuracy vs Model Complexity



but if instead we use a 3x3 network(details committed), which is a much less complex model, the accuracy percentage drops to 93.66% which satisfies claim made in equation 1. As all the curves are monotonically decreasing in the negative x-axis, having proved the fact for k=32, this claim will still hold of k=16,k=8..etc.

IV. Conclusion

Based on our experiments, we can conclude that given a baseline worst case test accuracy, we can systematically reduce the model complexity by following a structured approach and handling the trade-off between memory overhead and parameter required for tuning. For our experiment on MNIST[1] we reduced the parameters by 236 times, and memory footprint by 19.5 times as compared to the base model, and achieving the lower test accuracy threshold of 95% .

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. ”Gradient-based learning applied to document recognition.” Proceedings of the IEEE, 86(11):2278-2324, November 1998

[2] Martn Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Man, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Vigas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.