Size and Compression of Neural Networks

Naveen Tumkur Ramesh Babu
naveentumkurrameshbabu.1@osu.edu

James Enouen
enouen.8@osu.edu

Abstract -- This project was about exploring the size of a neural network and how it impacts the system's accuracy in classification. Primarily, the investigation focused on shrinking very large networks to fit smaller scale machines and on studying the accuracy of a model as its number of parameters grew larger and larger. In this first section, we assured that compression models could get very similar accuracies while using much fewer parameters. In the second section, we witnessed better testing accuracy and generalization if we continue to add parameters even if there is a period of time when we are perfectly fitting the training set. The overall conclusion we reached is that modern systems do not utilize their parameters to the greatest efficacy.

I. Introduction

This paper investigates how the size of a network impacts its accuracy and takes two primary perspectives to examine. The first is the side of compression, which takes a model and reduces the number of parameters while trying to maintain similar accuracy. One way to do this is by taking a trained model and pruning it to its 'essential' parameters. The other method, which we focus on, performs full training on the same task but with a model structure which is designed to use significantly less parameters than the original implementation (while maintain similar accuracy). The other perspective we take on network size is the asymptotics of when to stop adding parameters. In general, one may be unsure if adding extra parameters and complexity will be beneficial to their accuracy or if the training will end up being a massive waste of computational time by merely yielding worse generalization to their testing set. We believe it is not generally understood exactly where these thresholds lie, so the models people use are not able to fully take advantage of storage/ computation of their models. This paper begins to take a few steps starting to question how to resolve these issues and ultimately develop more potent neural networks.

II. Compression

There are two prior works which falls in the scope of our research [1][2]. "SqueezeNet [1]" uses smaller CNN which achieves AlexNet level accuracy on ImageNet dataset with 50x lesser parameters. Smaller models have several advantages: 1. They enable efficient distributed training 2. There is less overhead when exporting new models to clients 3. Feasible embedded systems deployment 4. Mobile Applications require very small sized models. For a fixed accuracy level, researchers explore several methods to reduce the model size. "Deep Compression [2]" explores methods to compress a deep neural network by using network pruning, quantization and Huffman coding. Using SqueezeNet and Deep Compression together yielded a model with less than 0.5 Mb size on ImageNet data set.

A. Methods

SqueezeNet uses three key strategies in order to substantially reduce the model size. Figure 1 explains SqueezeNet architecture. Firstly, SqueezeNet replaces 3x3 filters with 1x1 filters. This leads to 9x less parameters. Secondly, Squeeze Layer is used to reduce the number of input channels to 3x3 filters. Lastly, down sampling is done later in the network so that convolution layers would have large activation maps. Convolution Layer is replaced with a smaller Fire module. Fire Module is comprised of squeeze and expansion layer which helps in reducing the number of parameters. Also, fully connected layer is replaced with a softmax layer.

Deep Compression introduces techniques which can be used to substantially reduce the size of pretrained model. Deep Compression uses Network pruning to learn only important connections, the weights are quantized to allow weight sharing and Huffman coding is applied which uses smaller length bits to represent more frequently occurring weights. On an average, Pruning reduces the number of connections by 9x-13x, quantization and Huffman coding reduces the number of bits that represent each connection from 32 to 5.

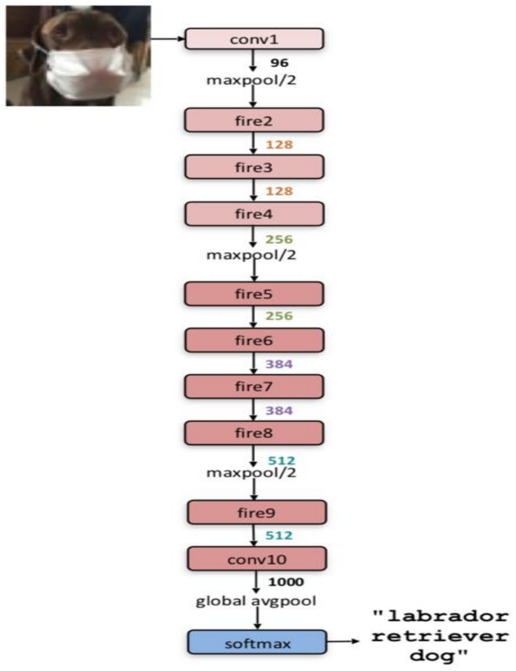


Figure 1: SqueezeNet Architecture

B. Experiments

Our plan is to first replicate the results of SqueezeNet paper, apply Deep Compression on obtained model. We plan to use Cifar-10 Dataset to understand the performance of compression techniques. Additionally, we plan to modify AlexNet architecture with few compression techniques we learned and measure performance [2] [3].

CNN Architecture	**Model Size**	**Compression Type**	**Data Type**	**Reduction in Model Size vs AlexNet**	**Top-1 ImageNet Accuracy**	**Top-5 ImageNet Accuracy**
AlexNet	240 MB	None	32 Bits	1X	57.2%	80.3%
AlexNet	240 MB -> 6.9 MB	Deep Compression	8 Bits	35X	57.2%	80.3%
SqueezeNet	4.8 MB	None	32 Bits	50X	57.5%	80.3%
SqueezeNet	4.8 MB -> 0.66 MB	Deep Compression	8 Bits	363X	57.5%	80.3%

Table 1: Compression techniques applied

Results from Table 1 shows that using SqueezeNet reduced model size of AlexNet (trained with ImageNet data) from 240 MB to 4.8 MB. Table 1 also shows applying Deep Compression directly on AlexNet reduced model size from 240 MB to 6.9 MB. We applied Deep Compression on obtained SqueezeNet model which further reduced model size from 4.8 MB to 0.66 MB. Deep Compression was applied with 8-bit quantization. The accuracy for top-1 and top-5 were same even after compression. SqueezeNet with Deep Compression reduced the model size by 363x compared to baseline (AlexNet without compression).

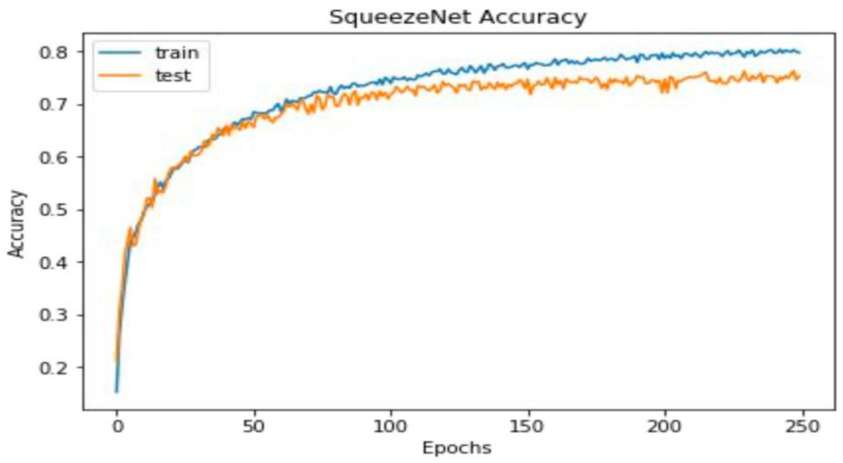
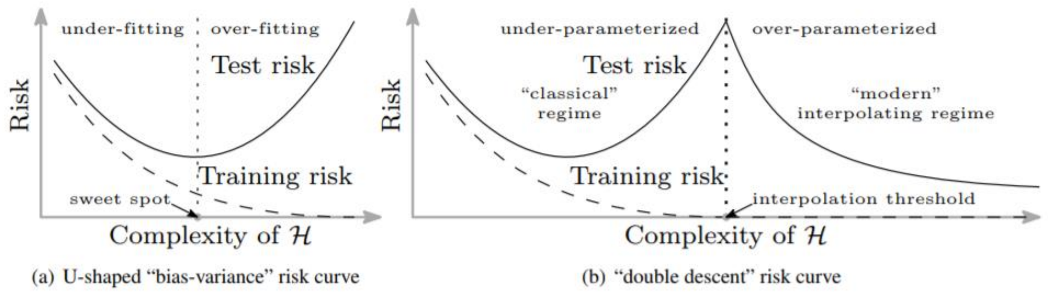


Figure 2: SqueezeNet on CIFAR-10 DataSet

We trained SqueezeNet with CIFAR-10 dataset. The experiment was conducted for 250 epochs and final model size was 1.6 MB. The baseline model size (AlexNet with CIFAR-10) was 190 MB. Applying Deep Compression to the obtained SqueezeNet model further reduced the model size to 0.9 MB. We conducted newer experiments: 1. In AlexNet Architecture we replaced fully connected layer with global average pooling and applied a softmax [5]. This didn't reduce the model size significantly because filter size was unchanged and down sampling was performed at earlier stage. 2. In AlexNet Architecture we replaced all the Convolution Layer with Fire Layer [5]. This reduced the model size from 190 MB to 27 MB without much compromise in accuracy.

III. Asymptotics

The central focus of this section about the investigation of asymptotics is the paper [6] which offers a new perspective on the classically believed bias-variance tradeoff. This idea suggests that we indeed get better results by adding more and more parameters/ hidden units to our model, despite the local maximum we approach as we reach the interpolation threshold of 'perfect fitting'. The paper suggests, moreover, that the minimum at the end of this curve tends to be lower than the local minimum attained in the bias-variance tradeoff curve.



This idea which they call the “double descent” curve is proven for Random Fourier Features, which is a model of a neural network with a single hidden layer where the first set of weights are fixed and the last set of weights are trained. The main issues that the paper suggests moving into actual neural networks are: there is a large variance between layers, making results difficult to replicate; random initialization of parameters additionally feed into this difficulty; and gradient descent can still find itself trapped at a local minima, rendering it unable to achieve the ‘second descent.’ Despite these challenges, we were also able to get reasonable reproduction of this curve on a neural network of our own.

A. Replication

The first step we took is to replicate the results achieved by the paper for a single hidden layer. We perform binary classification on a much smaller and simpler dataset than the paper because of computational limitations. Our dataset is a Gaussian distributed around the origin with a point’s class being determined by the quadrant in which it lies, as in Figure 3. This dataset is clearly not linearly separable, so we should not be able to perfectly classify this data with any type of linear classifier. Also, no convex classifier will be able to recognize this dataset as well. Hence, we look to see the double descent curve in action for this slightly abnormal scenario.

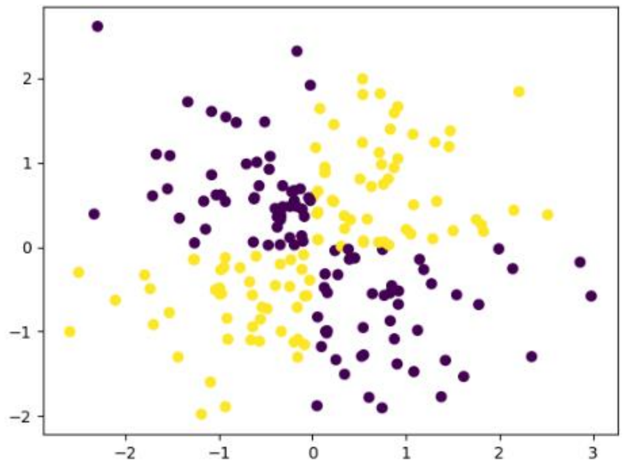
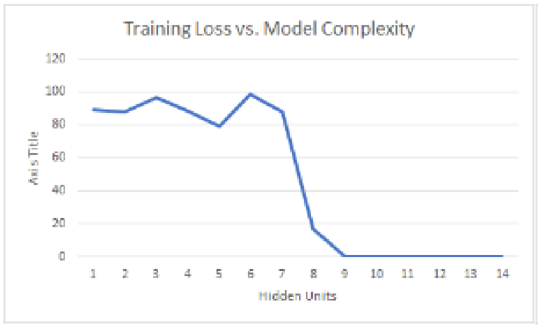


Figure 3: Binary classification dataset

Our results are also messier than the results the paper was able to achieve when applied to an actual neural network; however, the pattern is still apparent in our smaller set. We have that the training error dips to zero and as this occurs, there is a spike in the network’s validation error corresponding to the model’s lower generalizability. This is the idea preached by the “modern interpolation regime” [6] This provides more empirical evidence that we want to grow our models arbitrarily large in order to approximate our dataset arbitrarily well.



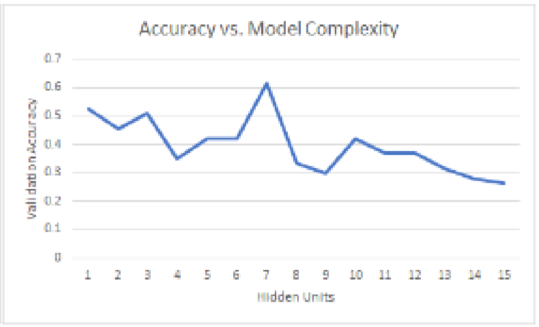
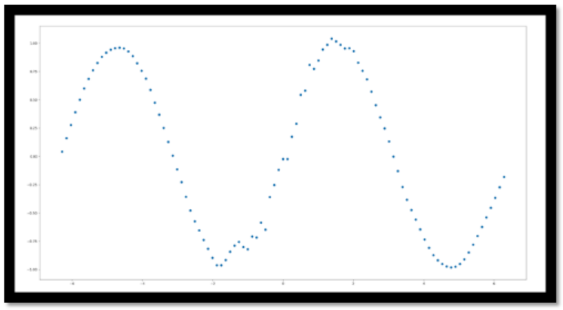


Figure 4: Training loss vs. model complexity

Figure 5: Testing accuracy vs. model complexity

B. Regression

We investigated how this idea would work when performing regression instead of classification. Because there is no type of regularization on our regression model, we expect that this method should simply fail. Overfitting in the traditional sense applies to the regression case and we get none of the benefits of overparameterizing our system. Our training loss is only a function of accuracy on the empirical values provided which causing the model to overfit to these particular points. We demonstrate this failure of the second descent by doing regression on a sine curve sampled in Figure 8. The wider hidden layer does not incentivize the network to learn a simpler curve, but rather throws unnecessary detail into the model since the model can already achieve zero loss on the testing set by directly fitting the examples. We highlight this by showing hand-picked results in Figure 6 and Figure 6 below for 10,000 hidden units and 100,000 hidden units where the lower parameterized model attains a much better approximation of the sine curve which illustrates the failure of overparameterization. This failure in a regression setting is a result which is probably understood by the writers of the paper; however, it is not mentioned in their paper, so we chose to highlight this failure of this second descent.



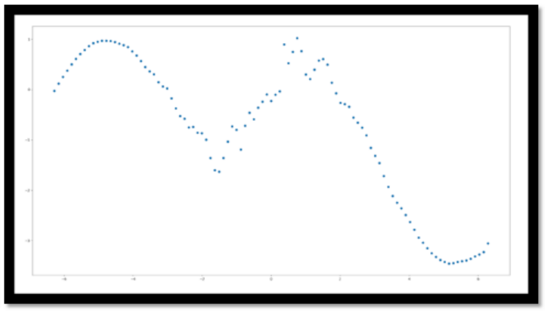


Figure 6: Predictions by 10,000 hidden units

Figure 7: Predictions by 100,000 hidden units

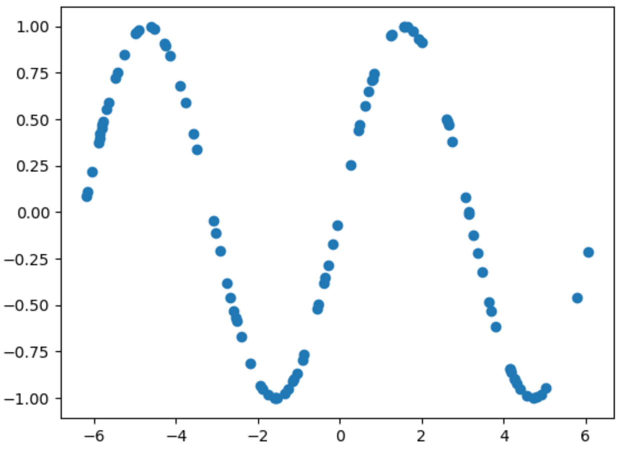


Figure 8: Regression classification dataset

C. Deep Neural Networks

Our largest step towards future work is investigating how the double descent curve shows up in a deep network setting. We still investigate binary classification, but instead of only looking at results from growing a single hidden layer's size, we also increase the number of layers used in the model. We trained on a dataset which is still relatively small but it is a slightly higher dimensional generalization of the first dataset we investigated for binary classification.

In our results, we see some remnants of the double descent curve, but it is not replicated in the way we originally anticipated. In Table 2 and Table 3 the results are displayed for hidden layer sizes between 1 and 20 and number of layers between 1 and 10. By looking at the first row, which corresponds to the single hidden layer, we observe the expected curve where the interpolation threshold spikes at 8 hidden units. This should and does work because it is exactly the double descent curve as in section A, Replication.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0.433333	0.366667	0.366667	0.433333	0.366667	0.4	0.466667	0.6	0.366667	0.3	0.3	0.366667	0.333333	0.433333	0.266667	0.3	0.366667	0.3	0.3	0.3
2	0.433333	0.4	0.5	0.366667	0.5	0.333333	0.5	0.466667	0.3	0.333333	0.366667	0.466667	0.466667	0.366667	0.3	0.333333	0.333333	0.3	0.266667	0.333333
3	0.433333	0.533333	0.366667	0.533333	0.5	0.333333	0.366667	0.3	0.366667	0.433333	0.466667	0.466667	0.366667	0.366667	0.333333	0.333333	0.333333	0.466667	0.333333	0.333333
4	0.433333	0.466667	0.466667	0.4	0.4	0.566667	0.466667	0.5	0.366667	0.5	0.5	0.333333	0.266667	0.333333	0.333333	0.4	0.3	0.433333	0.433333	0.366667
5	0.433333	0.466667	0.533333	0.633333	0.533333	0.5	0.5	0.3	0.533333	0.366667	0.433333	0.5	0.333333	0.3	0.266667	0.366667	0.366667	0.366667	0.333333	0.3
6	0.433333	0.433333	0.466667	0.533333	0.566667	0.533333	0.466667	0.533333	0.5	0.366667	0.466667	0.366667	0.3	0.433333	0.466667	0.333333	0.466667	0.366667	0.3	0.333333
7	0.433333	0.533333	0.533333	0.533333	0.466667	0.466667	0.466667	0.3	0.433333	0.333333	0.333333	0.4	0.533333	0.366667	0.266667	0.333333	0.233333	0.4	0.5	0.366667
8	0.433333	0.533333	0.533333	0.466667	0.466667	0.4	0.4	0.333333	0.466667	0.433333	0.466667	0.433333	0.366667	0.533333	0.466667	0.3	0.433333	0.3	0.2	0.433333
9	0.433333	0.533333	0.466667	0.466667	0.466667	0.633333	0.466667	0.466667	0.366667	0.466667	0.233333	0.4	0.366667	0.366667	0.533333	0.333333	0.4	0.466667	0.366667	0.333333
10	0.433333	0.533333	0.533333	0.533333	0.466667	0.5	0.366667	0.533333	0.533333	0.533333	0.433333	0.266667	0.4	0.366667	0.3	0.433333	0.4	0.333333	0.5	0.4

Table 2: Testing Accuracy for 1-10 hidden layers and 1-20 hidden units per layer

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	97.02632	90.4348	86.63248	72.07665	42.35226	53.41749	44.37948	16.85936	29.74843	0.044949	0.034171	0.039822	0.036969	0.036587	0.03199	0.040733	0.035384	0.042595	0.040032	0.035306
2	97.02632	84.55031	66.92546	69.87594	66.13527	66.16225	31.06688	78.4379	43.06353	0.016425	26.71854	0.01475	0.014481	0.013263	0.01103	0.012402	0.012258	0.012836	0.014031	0.014288
3	97.02631	97.0406	94.18376	97.0406	74.9623	28.28021	75.22748	45.88446	32.18212	57.36574	70.80814	0.001043	20.66171	9.300972	0.012431	0.004979	0.006055	0.00598	0.012817	0.006177
4	97.02632	97.0406	97.0406	71.41319	84.91762	78.45159	77.04355	81.5393	9.534114	54.11592	48.44199	0.002894	32.91599	0.003194	0.002747	1.912341	0.003607	0.003559	0.003613	0.01038
5	97.02631	97.0406	97.0406	81.49667	97.0406	93.48235	89.71216	71.05329	95.64008	61.0632	51.87159	91.25308	0.001854	0.001982	0.002471	0.001704	0.002098	0.001493	0.00277	0.002816
6	97.02631	96.82388	97.0406	97.0406	85.74022	97.0406	90.81361	94.96538	85.15954	13.46222	73.0162	0.001634	0.00153	0.000121	0.001267	0.0012	0.001279	0.00186	0.001168	0.001176
7	97.02632	97.0406	97.0406	97.0406	94.9742	97.0406	97.0406	22.73616	88.9465	34.1413	20.99095	62.31834	85.74213	0.001063	0.00074	0.001247	0.001153	0.001424	0.003981	0.000922
8	97.02632	97.0406	97.0406	97.0406	97.0406	72.30798	74.09555	77.90444	97.0406	58.70549	85.5654	92.09752	20.87354	94.20922	0.001418	0.000819	0.000638	0.00224	0.000533	0.000117
9	97.02632	97.0406	97.0406	97.0406	97.0406	94.65105	97.0406	97.0406	78.8308	97.0406	31.57683	0.001222	90.49293	5.92E-05	64.05565	0.001126	3.73E-05	0.000665	0.000573	0.000916
10	97.02632	97.0406	97.0406	97.0406	97.0406	89.46602	76.16321	97.04061	97.0406	97.0406	78.4537	39.61018	0.000312	0.000568	0.000868	3.140363	0.000565	30.14213	0.001077	0.000286

Table 3: Training loss for 1-10 hidden layers and 1-20 hidden units per layer

Regardless, we see the overall trend that these results disagree with the double descent curve as it is originally phrased. If the double descent curve was exactly a function of the complexity of the model we would see something closer to Figure 10 on the next page, but we actually see results more reflective of the other direction in Figure 9 below. This generally aligns with the viewpoint that a model loses a lot of stability near the interpolation threshold; however, we clearly need to do more investigation on the exact interplay this has with gradient descent and other methods to actualize these models. We do not fully understand why the models become so much more difficult to train despite the fact that they are strictly more general than their predecessors and should hence perform better. This is the most interesting discovery we made and certainly warrants more examination.

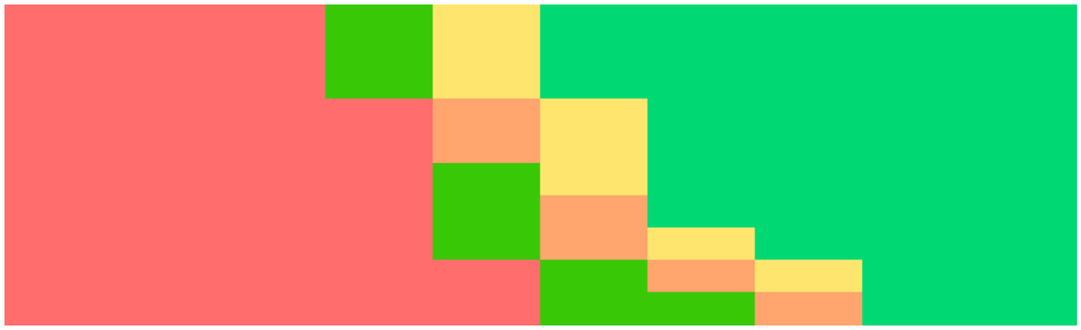


Figure 9: Witnessed pattern in testing error

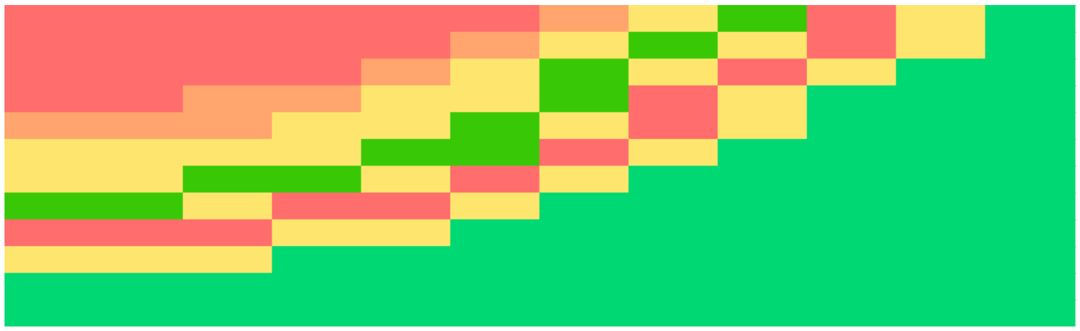


Figure 10: Expected pattern in testing error

IV. Conclusion

We understood methods to substantially reduce Neural Network model. We can choose an architecture which reduces number of parameters and apply compression techniques like pruning, quantization and Huffman encoding which greatly reduces model size. We can generalize compression to Microarchitecture compression which involves reducing the size of a module and Macroarchitecture approach which focuses on reducing the total size of model by modifying the architecture being used.

Our discovery is that the double descent curve is true and refines our previous beliefs for the bias-variance tradeoff. The fact that it did not perfectly reveal itself in a deeper network setting is concerning, but also very interesting because it indicates that the double descent curve cannot describe this phenomenon alone and there needs to be more work in understanding how gradient descent is able to train neural networks and the consequences on the generalizability of the trained model. As is generally understood, it is a little more difficult to train deeper networks (which could be the cause for our results) A suggested fix by the Swiss AI Lab in [7] is to add extra connections which span multiple layers to help information flow. This is one of many approaches which would compose a more thorough investigation of this topic.

Ultimately, this would lead to a fuller understand of when adding or removing parameters from our deep models yield more efficiency on the computer space and time that we are using. This would be very beneficial to the precise tuning of modern models.

Overall, we believe that deep neural networks need to be carefully studied to understand the required accuracy level. Having obtained required accuracy level, the model can be substantially compressed using an efficient Neural Network Architecture and employing different compression techniques. Model Compression is crucial for applications receiving frequent model updates and storage restricted devices like mobile, FPGA and embedded systems. Further study needs to be done to bring modern models the required tuning to fit smaller spaces and obtain greater accuracies.

V. References

[1] Forrest et. al, SqueezeNet: Alexnet-level accuracy with 50x fewer parameters and <1MB model size, CoRR, 2016, https://arxiv.org/abs/1602.07360

[2] Song et. al, Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman coding, ICPR, 2016, https://arxiv.org/abs/1510.00149

[3] Landola, Forest. SqueezeNet Repository. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters" https://github.com/DeepScale/SqueezeNet

[4] Sicong et. al, On-demand Deep Model Compression for Mobile Devices: Usage-Driven Model Selection F/W. https://www.tik.ee.ethz.ch/file/79a7dd6f6370f809e6180c0746232283/mobisys18-liu.pdf

[5] Yu Cheng et. al, A Survey of Model Compression and Acceleration for Deep Neural Networks, https://arxiv.org/pdf/1710.09282.pdf

[6] Belkin, M., Hsu, D., Ma, S., & Mandal, S. Reconciling modern machine learning and the bias-variance trade-off, 2018, https://arxiv.org/abs/1812.11118

[7] Srivastava, R. K., Schmidhuber, J., & Greff, K. Highway Networks, 2015, https://people.eecs.berkeley.edu/~brecht/papers/07.rah.rec.nips.pdf