CondenseNet : Reducing Parameters of CNN

A Report by

Sidharth Gulati

May 2, 2017

1 Introduction

Recent research on deep convolutional neural networks (CNN) has been primarily focused on improving accuracy. With a given level of accuracy, it is highly probable to identify multiple CNN architectures that achieve that accuracy level. But, CNN with reduced number of trainable parameters have at least two advantages: 1) They can be trained in less time and require less memory space. 2) They are more feasible for FPGA hardware with limited memory. This project explores the design space of the CNN architectures so that we can identify small CNN architectures. In this project, I propose a *CondenseNet* architecture that reduces the trainable parameters by almost 45x.

2 Experimental Setup

In this project, I used the MNIST dataset for training, validation and testing. This dataset contains 55,000 training images, 5,000 validation images and 10,000 testing images. The size of each image is 28×28. I used tensorflow for designing and implementing the deep neural network. The experiments are run on AWS (p2 instance) with Tesla K80 GPU.

3 Initial Model

The initial deep convolutional model contains 2 convolutional layer with max pooling and 2 fully connected layer.The architecture of initial model with number of parameters is described in Table 1. We can observe that the # of parameters to be trained in this initial model is in millions.

Now, in the next sections, I describe the variations in the architecture with the main goal of reducing # of parameters while preserving the accuracy level.

Table 1: Initial Model Architecture

Layer Name	**Filter Size/ Stride**	**# of Parameters**	**Memory (per Image)**
Input	-	-	28×28×1
conv-1	5×5/1	5×5×1×64 + 64	28×28×64
maxpool-1	2×2/2	-	14×14×64
conv-2	5×5/1	5×5×64×128 + 128	14×14×128
maxpool-2	2×2/2	-	7×7×128
FC-1	-	6272×512 + 512	512
FC-1	-	512×10 + 10	10
Total		**3,423,498**	**0.38MB**

4 Motivation and Strategy

In order to reduce the number of parameters in a convolutional neural network, we first have to observe how the # of parameters are calculated in a convolutional layer. Consider a convolutional layer, l_i, in a deep neural network that is comprised of $k \times k$ filters. The total number of parameters $s(l_i)$ in this layer is given below :

$$s(l_i) = k \times k \times \text{num of filters} \times \text{input channels}$$

Now, to decrease $s(l_i)$, we can follow below three strategies:

Strategy 1. Reduce the filter size. Given, a particular image size, we can decrease the filter size, that is, k which would result in a decrease in the # of parameters required to train the deep convolutional network.

Strategy 2. Decrease the number of of input channels. Given, a particular image size, we can decrease the input channels. But, in the given design space, we already have only one input channel. So, this strategy is not applicable for this particular dataset.

Strategy 3. Downsample number of filters that is, reduce number of feature maps. This approach also decreases the number of trainable parameters but, reducing feature maps would also result in decrease in accuracy. So, we have to compensate for the decrease in the decrease of accuracy. This can be done, by first reducing the number of filters and then increasing them in the later layers. I hypothesize that larger number of feature maps in the later layers of a deep convolutional network will result in higher accuracy. This hypothesis is motivated by the fact that as we go from lower to higher layers, CNN learns from simple to complex features. The complex features are more important in classifying the images as we reach near the classification layer. We can also visualize this with an example from this dataset. In MNIST, the task is to classify digits from the images and as we go towards the later layers, we form the images from the extracted features that closely resemble the input image. Using this approach, I propose the *CondenseNet* model that is described in later sections.

5 Proposed Models

5.1 Modification 1 and Modification 2 (Using Strategy 1.)

As stated in Strategy 1, reducing the filter size reduces the number of parameters. In Modification 1 and Modification 2 models, instead of using 5×5 filters in the convolutional layers, I use 3×3 and 1×1 respectively. The number of parameters and memory footprint for Modification 1 and Modification 2 model is describer in Table 2 and Table 3 respectively.

Table 2: Modification 1

Layer Name	Filter Size/ Stride	# of Parameters	Memory (per Image)
Input	-	-	28×28×1
conv-1	3×3/1	3×3×1×64 + 64	28×28×64
maxpool-1	2×2/2	-	14×14×64
conv-2	3×3/1	3×3×64×128 + 128	14×14×128
maxpool-2	2×2/2	-	7×7×128
FC-1	-	6272×512 + 512	512
FC-1	-	512×10 + 10	10
Total		**3,291,402**	**0.38MB**

Table 3: Modification 2

Layer Name	Filter Size/ Stride	# of Parameters	Memory (per Image)
Input	-	-	28×28×1
conv-1	1×1/1	1×1×1×64 + 64	28×28×64
maxpool-1	2×2/2	-	14×14×64
conv-2	1×1/1	1×1×64×128 + 128	14×14×128
maxpool-2	2×2/2	-	7×7×128
FC-1	-	6272×512 + 512	512
FC-1	-	512×10 + 10	10
Total		**3,225,354**	**0.38MB**

The accuracy for Modification 1 and Modification 2 models is tabulated in Table 4.

Table 4: Accuracy Statistics

Model	Testing Accuracy
Modification 1	98.75%
Modification 2	94.37%

5.2 CondenseNet (Strategy 1 + Strategy 3)

The motivation of CondenseNet is the existing Inception architecture [1] where it uses a combination of 3×3 and 1×1 filters. As stated in Strategy 3, reducing number of feature maps, may decrease the accuracy. To compensate this, I use a cascade module, namely *condensed_module*, of *condense-expand* layers. The main idea is to first condense the convolutional layers using 1×1 filters. Then the output of this condensed layer is given to the *expand* layer, which uses a combination of 3×3 and 1×1 filters to increase the number of activation maps.

Let $\mathbf{c_{1\times1}}$ be the number of 1×1 filters in the *condense* layer and $\mathbf{e_{1\times1}}$ and $\mathbf{e_{3\times3}}$ be the number of 1×1 and 3×3 filters in the *expand* layer. In *CondenseNet*, I make sure that, $\mathbf{c_{1\times1}} < \mathbf{e_{3\times3}} + \mathbf{e_{1\times1}}$ as we are condensing and expanding sequentially. We are increasing the number of feature maps later, because the increase would compensate the decrease of accuracy introduced by *condense* layer. Note, that the *condensed_module* is used in place of traditional convolutional layers in the deep convolutional model.

Moreover, in all *CondenseNet* models (vanilla + modifications), $\mathbf{e_{3\times3}} < \mathbf{e_{1\times1}}$ in the *expand* layer because, the parameters are decreased by 9X when the said condition is satisfied. This, results in the decrease of parameters considerably.

I experimented with different modifications in the *CondenseNet* and the number of parameters and memory footprint of each modification is given below.

5.2.1 Vanilla CondenseNet

The number of parameters and memory footprint of the *Vanilla CondenseNet* is tabulated in Table 5. In vanilla CondenseNet, $\mathbf{c_{1\times1}} = \mathbf{16}$, $\mathbf{e_{3\times3}} = \mathbf{32}$ and $\mathbf{e_{1\times1}} = \mathbf{96}$.

Table 5: Vanilla CondenseNet

Layer Name	Filter Size/ Stride	# of Parameters	Memory (per Image)
Input	-	-	28×28×1
conv-1	3×3/1	3×3×1×64 + 64	28×28×64
maxpool-1	2×2/2	-	14×14×64
condense-1	1×1/1	1×1×64×16 + 16	14×14×16
expand-1	1×1,3×3/1	1×1×16×96 + 3×3×16×32 + 128	14×14×128
maxpool-2	4×4/4	-	4×4×128
FC-1	-	2048×256 + 256	256
FC-1	-	256×10 + 10	10
Total		**535,066**	**0.375MB**

5.2.2 CondenseNet-1

In CondenseNet-1, the parameters of condense module is same as Vanilla CondenseNet. I modified the strides of the maxpool-2 layer which further resulted in the decrease of number of parameters. The number of parameters and memory footprint is given in Table 6.

Table 6: CondenseNet-1

Layer Name	Filter Size/ Stride	# of Parameters	Memory (per Image)
Input	-	-	28×28×1
conv-1	3×3/1	3×3×1×64 + 64	28×28×64
maxpool-1	2×2/2	-	14×14×64
condense-1	1×1/1	1×1×64×16 + 16	14×14×16
expand-1	1×1,3×3/1	1×1×16×96 + 3×3×16×32 + 128	14×14×128
maxpool-2	7×7/7	-	2×2×128
FC-1	-	512×256 + 256	256
FC-1	-	256×10 + 10	10
Total		**141,850**	**0.37MB**

5.2.3 CondenseNet-2

Similarly, I decreased the output of FC1 layer in CondenseNet-2 with stride length same as CondenseNet-1. The number of parameters and memory footprint is given in Table 7.

Table 7: CondenseNet - 2

Layer Name	Filter Size/ Stride	# of Parameters	Memory (per Image)
Input	-	-	28×28×1
conv-1	3×3/1	3×3×1×64 + 64	28×28×64
maxpool-1	2×2/2	-	14×14×64
condense-1	1×1/1	1×1×64×16 + 16	14×14×16
expand-1	1×1,3×3/1	1×1×16×96 + 3×3×16×32 + 128	14×14×128
maxpool-2	7×7/7	-	2×2×128
FC-1	-	512×128 + 128	128
FC-1	-	128×10 + 10	10
Total		**74,906**	**0.368MB**

5.2.4 CondenseNet-3

Similarly, I decreased the output of FC1 layer in CondenseNet-3 keeping strides length = 4 in maxpool-2. The number of parameters and memory footprint is given in Table 8.

Table 8: CondenseNet - 3

Layer Name	Filter Size/ Stride	# of Parameters	Memory (per Image)
Input	-	-	28×28×1
conv-1	3×3/1	3×3×1×64 + 64	28×28×64
maxpool-1	2×2/2	-	14×14×64
condense-1	1×1/1	1×1×64×16 + 16	14×14×16
expand-1	1×1,3×3 /1	1×1×16×96 + 3×3×16×32 + 128	14×14×128
maxpool-2	4×4/4	-	4×4×128
FC-1	-	2048×64 + 64	64
FC-1	-	64×10 + 10	10
Total		**139,738**	**0.374MB**

5.3 Performance

The accuracy for all *CondenseNet* models (vanilla + modifications) is tabulated in Table 9.

Table 9: Accuracy Statistics of CondenseNet

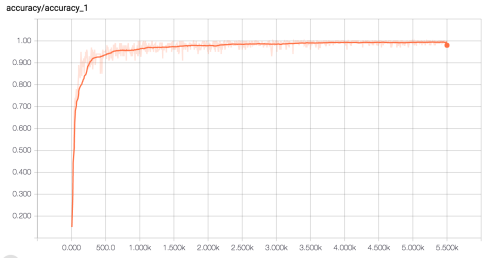
Model	Testing Accuracy
Vanilla CondenseNet	98.75%
CondenseNet-1	97.6%
CondenseNet-2	97.3%
CondenseNet-3	97.7%

6 Results

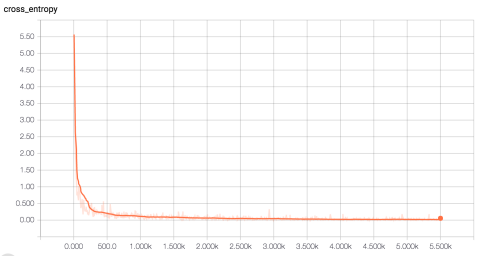
The performance graph references for models described above in tabulated in Table 10.

Table 10: Performance Graph References

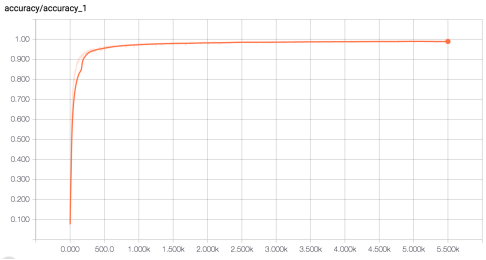
Model	References
Initial Model	Figure 1
Modification 1	Figure 2
Modification 2	Figure 3
Vanilla CondenseNet	Figure 4
CondenseNet-1	Figure 5
CondenseNet-2	Figure 6
CondenseNet-3	Figure 7



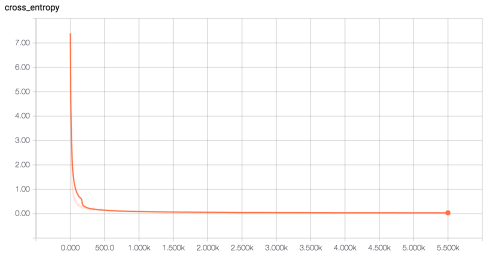
(a) **Training Accuracy**



(b) **Training Cross Entropy**

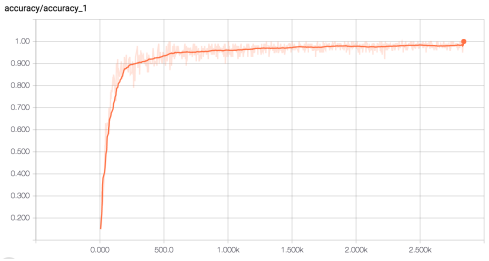


(c) **Validation Accuracy**

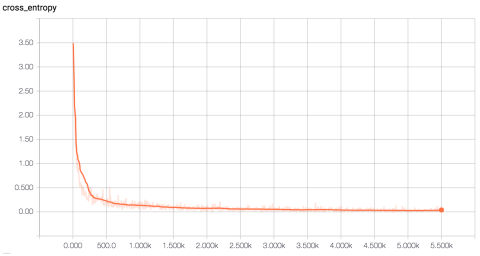


(d) **Validation Cross Entropy**

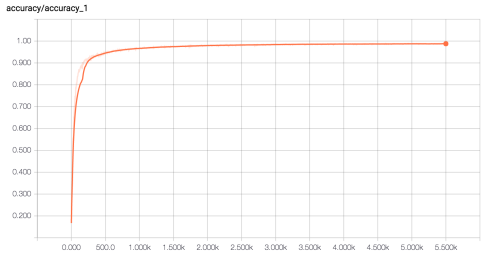
Figure 1: Performance Graphs for Initial Model



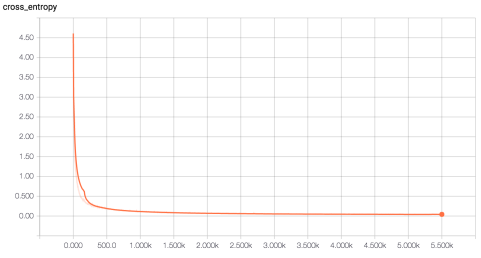
(a) **Training Accuracy**



(b) **Training Cross Entropy**

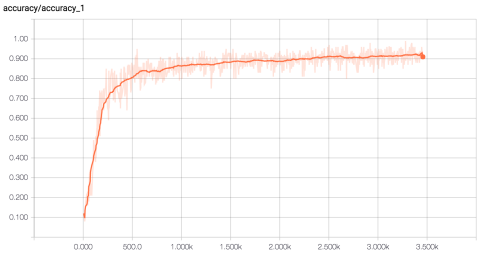


(c) **Validation Accuracy**

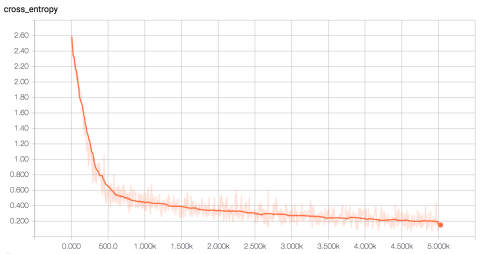


(d) **Validation Cross Entropy**

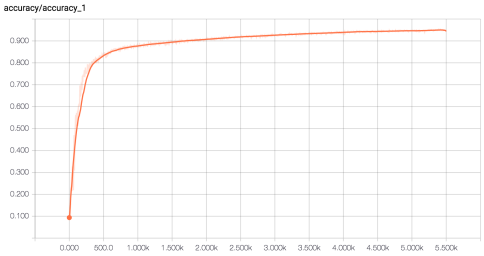
Figure 2: Performance Graphs for Modification 1



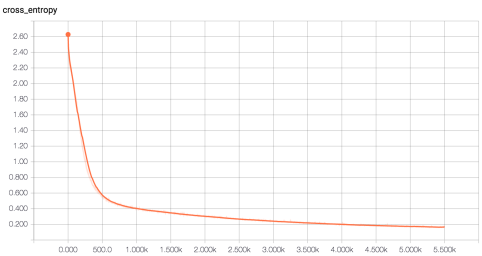
(a) **Training Accuracy**



(b) **Training Cross Entropy**

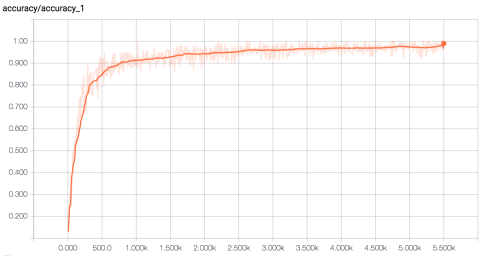


(c) **Validation Accuracy**

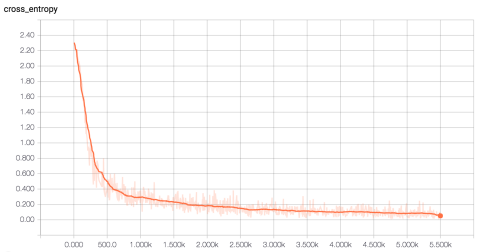


(d) **Validation Cross Entropy**

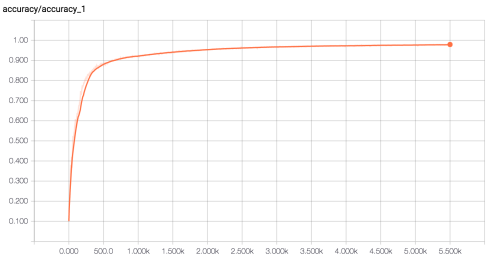
Figure 3: Performance Graphs for Modification 2



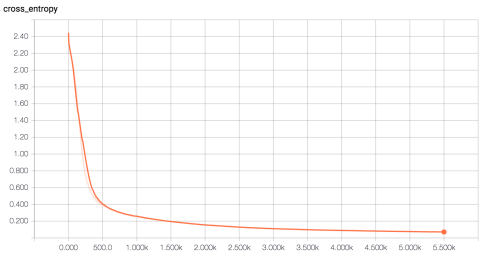
(a) **Training Accuracy**



(b) **Training Cross Entropy**

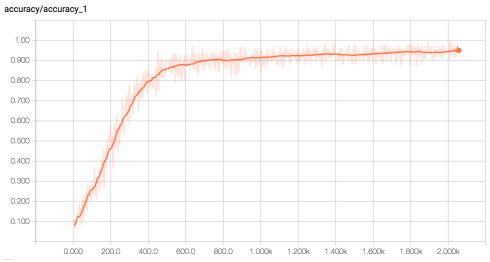


(c) **Validation Accuracy**

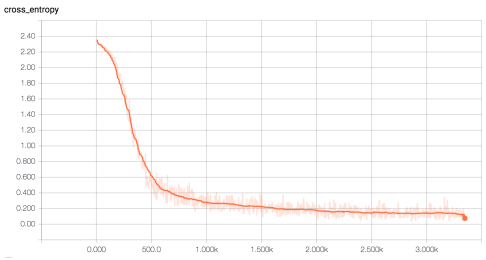


(d) **Validation Cross Entropy**

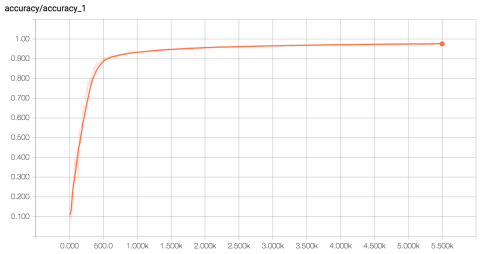
Figure 4: Performance Graphs for Vanilla CondenseNet



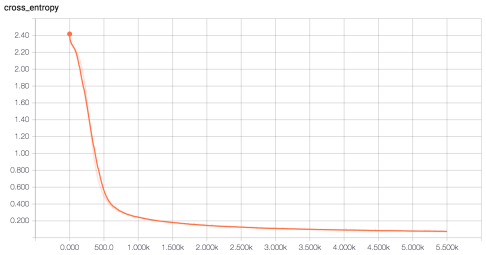
(a) **Training Accuracy**



(b) **Training Cross Entropy**

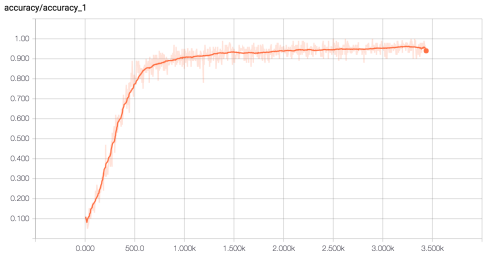


(c) **Validation Accuracy**

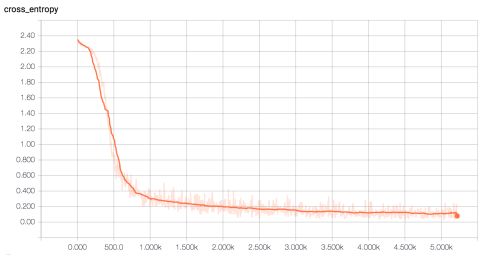


(d) **Validation Cross Entropy**

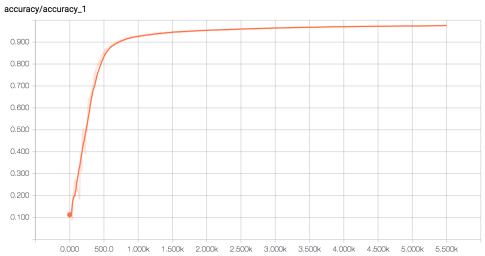
Figure 5: Performance Graphs for CondenseNet-1



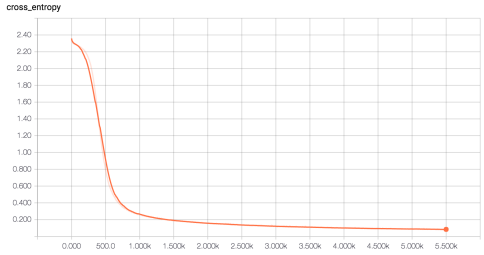
(a) **Training Accuracy**



(b) **Training Cross Entropy**

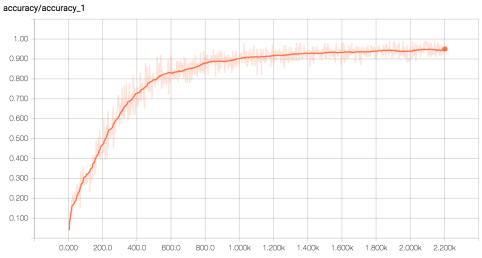


(c) **Validation Accuracy**

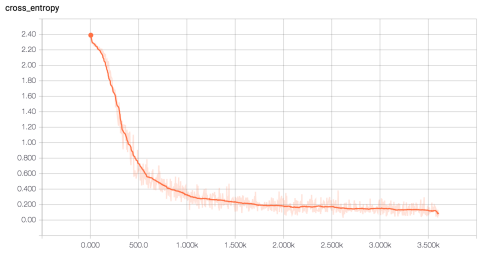


(d) **Validation Cross Entropy**

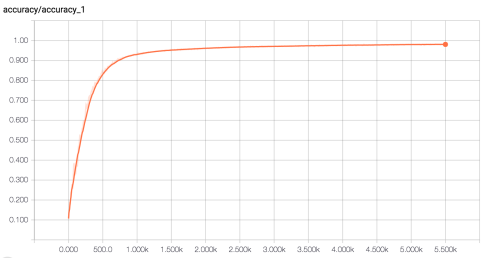
Figure 6: Performance Graphs for CondenseNet-2



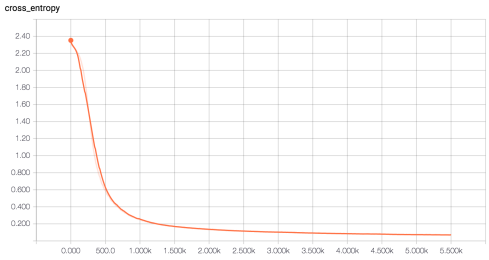
(a) **Training Accuracy**



(b) **Training Cross Entropy**



(c) **Validation Accuracy**



(d) **Validation Cross Entropy**

Figure 7: Performance Graphs for CondenseNet-3

The comparison of number of parameters and accuracy with Initial Model is given in Table 11.

Table 11: # of Parameter Statistics of CondenseNet compared with Initial Model

Model	Accuracy	# of Parameters	Decrease in # of params
Initial Model	99.05 %	3,423,498	-
Modification 1	98.75 %	3,291,402	1.04x
Modification 2	94.37 %	3,225,354	1.06x
Vanilla CondenseNet	97.73 %	535,066	6x
CondenseNet-1	97.6 %	141,850	24x
CondenseNet-2	97.38 %	74,906	45x
CondenseNet-3	97.7 %	139,738	24x

The training time comparison for the models described is shown in Table 12.

As you can from Table 11, I was able to reduce the parameters upto **45x** with an accuracy decrease of **1.67%** when compared with the initial model. From the above table, **CondenseNet-2** performed the best when compared with the decrease in number of parameters and accuracy stability. Also, the training time for *CondenseNet* (Vanilla + Modifications) is smaller as compared to Initial Model and Modification 1 and Modification 2.

Table 12: Training Time Statistics

Model	Training Time
Initial Model	0.51 secs
Modification 1	0.4 secs
Modification 2	0.35 secs
Vanilla CondenseNet	0.16 secs
CondenseNet-1	0.146 secs
CondenseNet-2	0.143 secs
CondenseNet-3	0.145 secs

7 Does Reduced Parameters Corresponds to Low Accuracy?

For the above question, I couldn't quantify the relation between number of parameters and accuracy in general. This is due to the fact, having large number of parameters doesn't imply an increase in the accuracy. The question doesn't take overfitting into account. With large number of parameters, we risk introducing over fitting and this will decrease the accuracy in the test dataset. In this design space, we observe that though the number of parameters in Vanilla CondenseNet is lower than Modification 2 model, accuracy of Vanilla CondenseNet is higher than the Modification 2 model. This is in direct contradiction to the claim made in the above question.

8 Conclusion

This project possed an interesting problem where I was able to explore the design space of deep convolutional neural networks. From the above results, it is clear that *CondenseNet* has the highest performance with the given limitations. Though, the dataset was very small and there was not enough room to play with other parameters, I hypothesize that *CondenseNet* will definitely perform well with less number of parameters in other deep convolutional neural network architectures like Inception, Alexnet, VGGNet etc. This hypothesis is yet to be tested. Also, I tried using SVD to reduce the rank of weights in the convolutional and fully connected layers, but this method is not feasible for deep neural networks which require training from scratch. If, we have a pre-trained network, then we can utilize Low Rank Approximation of weight matrices and train on them to have a reduce memory utilization and speed-up training.

9 Future Work

In future, I would like to test the *CondenseNet* on CiFAR10 and ImageNet dataset so that we can observe how it performs with large datasets and appreciate the architecture with greater number of condensed modules. I would also like to try SVD Low Rank Approximation

technique on a pre-trained network.

References

[1] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.