COMP 551 S001-Ablation Study

Densely Connected Convolutional Neural Networks

RARI 458

James Brace
McGill University
james.brace@mail.mcgill.ca
260654858

Nima Adibpour
McGill University
nima.adibpour@mail.mcgill.ca
260606511

Paul-Andre Henegar
McGill University
paul-andre.henegar@mail.mcgill.ca
260689725

April 21, 2018

Abstract

In this paper we dissect the internals of the current state-of-the-art convolutional network, known as DenseNet. Using the CIFAR-10 dataset, we tweak the underlying architecture and its parameters to gain insight on the model's robustness, regularization effect, and potential ways to improve performance. The code for our study can be found in `https://github.com/nima200/densenet-ablation`

I. Introduction

One of the current trends in Machine Learning academia is to create state-of-the-art Convolutional Neural Networks (CNN) that perform well on large datasets. This race for performance was perhaps made popular by the annual ILSVRC competition which brought competitive Machine Learning to a large-scale. Now that we are in an era where we can accurately classify large datasets such as ImageNet, some researchers explore CNNs in order to produce the latest standard performance of accuracy, but with reduced training times and smaller models. One of the most recent spawns of this CNN movement is DenseNet. DenseNet aims to tackle a wide-range of challenges present during CNN development not by implementing some fancy tricks such as compression or stochastic depth, but rather by incorporating a very unique architecture that preserves data flow as the model gets into its deepest layers. The idea behind DenseNet is to feed the output of any convolutional layer to all subsequent layers by concatenating the output features with each input. This aims at reducing the effects of the vanishing gradient problem as well as taking advantage of feature reuse, which perhaps counter-intuitively, reduces the number of weights being trained on. This paper is meant to explore the robustness of this model by making changes to the architecture and its hyper-parameters, as well as exploring some of the claims made in the paper. Throughout this ablatation study, we also hope to provide insights for improved performance.

II. Background

i. DenseNet

Architecure DenseNet's architecture stems from observations made on ResNet's connection-skipping via identity functions. It was observed that the advantage of this method was to allow the gradient to flow from early layers to later layers. However, the researchers felt that the summation that ResNet was performing resulted in a loss of valuable information. DenseNet aims to capitalize on gradient flow, not by summation, but by concatenation. This is done by introducing direct connections from any layer to all subsequent layers. Of course, the concatenation operation is not viable when the size of feature-maps changes. This conflicts with the need of CNNs to down-sample layers through pooling. In order to work around this Huang et al. introduced the concept of *dense blocks* and *transition layers*. The idea of *dense blocks* is to perform the composite functions that maintain the feature-map size and to feed the output to the rest of the layers, whereas the *transition layer* performs average-pooling in order to reduce the feature-map size. Their architecture introduces a hyper-parameter unique to this model, the *growth-rate*. The *growth-rate* regulates how much new information each layer

contributes to the cumulative feature-map, which they call the global state. The global state, once written, can be accessed from everywhere within the network, and unlike in traditional network architectures, there is no need to replicate it from layer to layer. In addition to all of these layers, they also introduce *bottleneck layers* in between convolutional layers and *compression layers* in between dense blocks to reduce the number of input feature-maps and thus increase computational efficiency. The bottleneck layer is essentially just a 1x1 convolutional layer. The visual representation of DenseNet can be seen in **Figure 11**. **Experiments** For their experiments, they created various versions of their models with different growth rates and with the option to have the bottleneck layer discussed before. In addition to varying models, they also tested on different datasets: CIFAR-10, CIFAR-10+, CIFAR-100, CIFAR-100+, ImageNet, and SVHN. The '+' represents a dataset with affine transformations applied to the images for data augmentation. They then trained on these datasets using stochastic gradient descent with learning rate scheduling.
Results and Conclusions Needless to say, they were very happy with their outcomes. DenseNet outperformed all of the latest state-of-the-art models (VGG, Highway, and ResNet) on all datasets. They were able to beat ResNet's accuracy with 30x less parameters as well. They observed that their deeper models performed the best which shows that their model does not suffer from over-fitting or optimization difficulties present in ResNet. They also reported a regularization effect that they believe is caused by the growth factor and the bottleneck layer, which we will explore in our ablation.

III. Ablation Methodology

For our ablation study, we decided to look at three components of the model: the architecture, the hyper-parameters, and the performance. For dissecting the architecture we are going to change the core component of the model and observe the effects it has on the performance. We have decided to minimize the efforts on tuning the original hyper-parameters reported by the paper, due to the fact that if the paper claims the existence of a hyper-parameter, it is most likely that they have tuned it as well and there are more interesting changes to be made to the architecture than reproduce the same ideal hyper-parameters that they have picked. Lastly, we want to question some of the claims made in paper about performance and see to what extent they hold true. For all our studies we have primarily ran the results on the CIFAR-10 dataset with data augmentation enabled, with a depth of 40 layers in the model. We have used a Keras [2] implementation [3] of DenseNet as the base of all ablations.

i. Architecture Experiments

For our architecture experiments we are going to look at the following:

- The effects of only using feature maps from n of the previous layers instead of all of them, where the value n would now be a new hyper-parameter.
- Dropout layers before and after convolutional blocks to observe regularizing effects on performance.
- Swap out average pool with max pool and observe the effect on multiple datasets to see if such a choice is architecture dependent or dataset dependent.
- Change the order of operations for the composite function represented by the dense blocks. Right now it exists as Batch Normalization, followed by a ReLU and a 3x3 convolutional layer.
- Change the number of dense blocks and observe the effects on performance with different datasets.
- Make the growth rate vary within each block, i.e. make the growth rate larger on earlier layers and smaller on later layers or the other way around.

ii. Hyper-parameter Experiments

For our hyper-parameter experiments we are going to play around with following:

- The growth rate coefficient in bottleneck layers.

iii. Performance Study

To measure how robust the outcomes of DenseNet are we are going to observe and explain the following:

- The regularization effect claimed to be caused by the growth rate and the bottleneck layers
- How the model performs on simpler datasets, i.e. ones that may lead to more over-fitting given the complexity of the model. An example of a dataset we will use in Fashion MNIST.

IV. Ablation Results

i. Architecture Experiments

i.1 Use n Previous Layers

From the original paper, there is a graph, shown in **Figure 12**, displaying the average absolute value of weights in order to convey how much data flows from one layer to another.

It was noticed that as you get deeper into the network, even though they still use weights across the entire block, the final layers seem to concentrate more on final feature-maps. This brings up the question of whether some connections can be seen as redundant, and perhaps dropped. This motivated us to test whether all connections are necessary or if it would be enough to only create connections to the previous, say, N layers.

We tested with various values of N on a 3-block 40-layer DenseNet with no bottleneck layers and 12 3x3 convolution layers per block. The dataset we used was non-augmented CIFAR-10. We compare the results with the result of the same setup but with the block being fully connected (which would be equivalent to $N = 12$). Results can be seen in **Figure 1** and in **Table 1**.

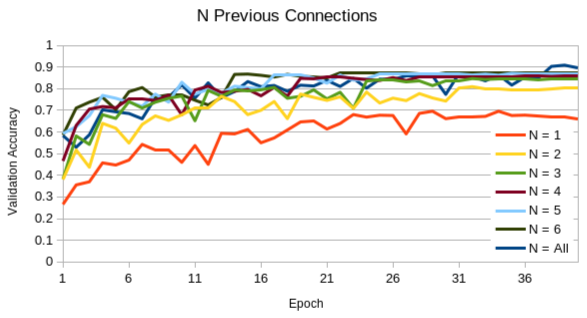

Figure 1: *Performance of varying number N of previous layers used*

The training graphs indicate what we would expect: having more connections to previous layers improves performance. However, the improvement becomes smaller as n becomes larger. Even though the results aren't as good as with a fully connected block, they come close. However, as seen in **Table 1**, the training time per epoch for the case when N=6 is not much better compared to the case when N=All. This can however be an issue of our implementation in Keras since we did not implement the concatenation of the N previous feature-maps in the same way we implemented the concatenation for the fully connected block.

It was observed that the validation accuracy stabilizes for values of N that sit between 3 and 6. This could suggest that taking the last N layers adds an effect of regularization while still keeping the model simple. It's also possible that the N=All version would have also stabilized if we let it run for long enough. There's also a noticeable trend that as N increases, the number of epochs at which the stabilization starts decreases, but for some reason this does not happen for N=All. A possible next step would be to explore this for all values of N in between 0 and All=12.

i.2 Additional Dropout Layers

The motivation behind this study was to observe if we can further reduce over-fitting. Although we have not observed any cases of severe over-fitting with this network, we believe it's sensible to test for whether dropout layers can help test performance. As a result, we designed dropout layers to go in two specific locations in each convolutional block: before, and after each block. We also observed the effects of adding dropout after concatenation of features, but that dropped performance by large margins due to the fact that a lot of previous connections would be dropped. This in fact defeats the purpose of having a densely connected network. The results are shown in **Figure 2** (after) and in **Figure 3** (before).

In both cases we observe similar performance through training yet lower performance at the very end where a scheduled learning rate update for 0% dropout (which is what the paper has) results significantly better performance than adding a dropout layer. These results were somewhat expected yet we still wanted to investigate the real effect and it is now proven to us that the dense connections

	1	2	3	4	5	6	All
Accuracy	69.62%	80.96%	84.69%	85.82%	87.14%	87.57%	90.89%
Error	30.38%	19.04%	15.31%	14.18%	12.86%	12.43%	9.11%
Params	107,482	210,274	309,034	403,762	494,558	581,122	1,078,018
Epoch Time	84s	118s	161s	184s	212s	247s	264s

Table 1: *Numerical results of varying number N of previous layers used*

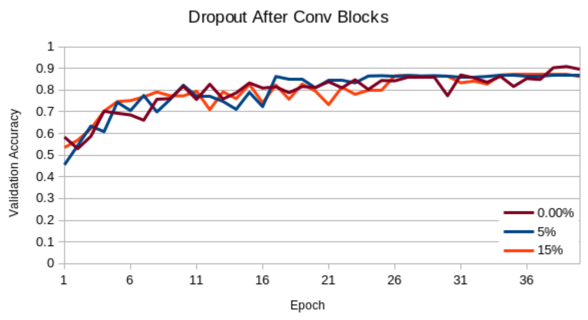

Figure 2: *Performance with varying dropout percentage added after convolutional blocks*

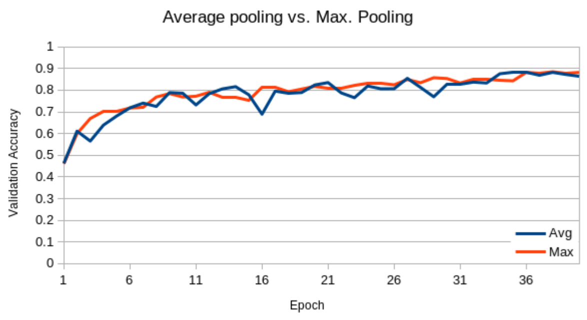

Figure 4: *Performance of Average Pooling vs Max Pooling*

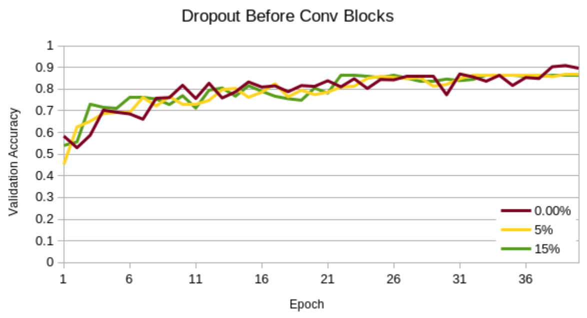

Figure 3: *Performance with varying dropout percentage added before convolutional blocks*

alone have a very good regularizing effect that prevents any over-fitting from happening. Due to the decreased final performance of the model with dropout layers added, we decided not to experiment with higher percentages of dropout as the effect would get worse.

i.3 Average Pooling vs Max Pooling

It was noticed that the architecture used Average Pooling for all of its pooling layers. Having little knowledge of the effects of Average Pooling vs Max Pooling we decided to swap them out and see the outcome. The results can be seen in **Figure 4**

It's worth noting that changing to Max Pooling changed to overall classification layer since the classification layer relies on a Global Average Pooling operation. In literature, the benefit of this is that it is more native to the convolution structure by enforcing correspondences between feature maps and categories. Thus the feature maps can be easily interpreted as categories confidence maps. Another advantage is that there is no parameter to optimize in the global average pooling thus overfitting is avoided at this layer [5]. Given that, it can be summarized that Max Pooling picks out more extreme features, whereas Average Pooling takes all features into account.

The results went against our intuition; Max Pooling performed better than Average Pooling by a very slight margin. Max Pooling also resulted in a much smoother accuracy curve. This could be due to the fact that Max Pooling will pick out the extreme features from the beginning and further results will be biased towards those initial selections. On the other hand, average pooling will always take into consideration all the features, so irregularities during training will have a much larger effect on the accuracy per epoch.

i.4 Various Combinations of Composite Layers

i.5 Effect of Bottleneck Layers

The authors claimed that the added bottleneck layers that would reduce the feature-map from all previous layers to a feature-map of fixed size before performing the 3x3 convolution decreased the number of features and increased computational efficiency. We explored the validity of this claim for small networks.

We compared a 3-block DenseNet where each block 6 bottleneck layers and 6 convolution layers with a DenseNet where each block had 12 convolution layers and with a DenseNet where each block had 6 convolution layers and no bottleneck layers. All of these experiments were carried on augmented CIFAR-10. Results can be seen in **Figure 5** and in **Table 2**.

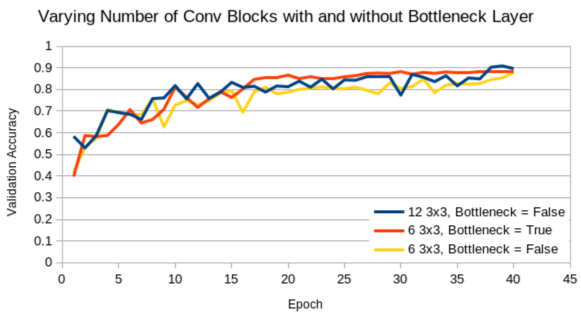

Figure 5: *Performance of removing bottleneck layers*

As expected, the network with 12 convolutional layers per block gave the best accuracy. Out of the two networks with the same amount of convolutional layers, the one with bottleneck layers performed better. This would indicate that the bottleneck layers don't just compress information but actually perform computations on their own, or at least that's the case in not very deep networks where every layer helps.

It turns out that even though there is slightly more parameters for the DenseNet with 6 convolutions per block and no bottleneck layers, it is faster to train. This might be because even though there are more parameters, since there are less layers, running the neural network is more parallelizable on the GPU.

i.6 Number of Dense Blocks

Throughout the paper, there are no references as to whether the number of dense blocks in the network is a hyper-parameter that can be tuned or whether the default 3 dense blocks that they have used for all their experiments is a core part of the architecture. As a result of this, we decided that it would be a valid study to observe how the number of dense blocks affect the performance, and whether this performance change is dataset specific which would then make it be considered a hyper-parameter of the model, or if it is a fixed value that performs best for this architecture. We start by tuning the number of dense blocks on CIFAR-10. The results are shown in **Figure 6**.

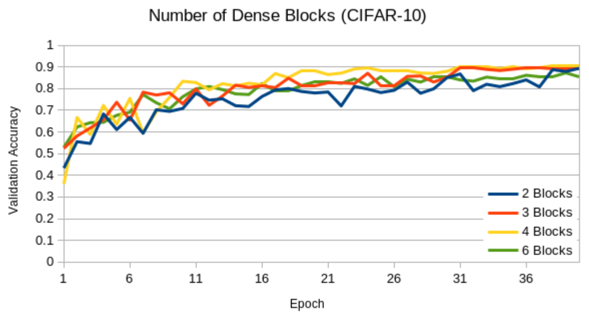

Figure 6: *Performance of varying number of dense blocks on CIFAR-10 dataset*

From the graph we observe that changes in the number of dense blocks does affect the test performance, but not by a large margin. In the case with CIFAR-10, 2 blocks performs the worst and 4 blocks achieves better performance much earlier into the training than 3 blocks which the paper has decided to go with. However we can also observe that an increase beyond 4 does has a hit on performance as well. To further study this, we consider the effects of varying number of dense blocks on the SVHN dataset. The results are shown in **Figure 7**.
On SVHN, we observe a similar but slightly different situation. We observe that 4 dense blocks does reach higher validation accuracy earlier in training as in CIFAR-10, however 3 blocks give a slightly better performance by a fraction of a percentage (3 blocks = 94.49% vs. 4 blocks = 94.39%). Although the difference in performance between the two configurations are very small, from this we conclude that the number of dense blocks for the network can

	6 Conv., 6 Bottleneck	12 Conv.	6 Conv.
Accuracy	88.33%	90.89%	87.85%
Error	11.67%	9.11%	12.15%
Params	257,218	1,078,018	296,530
Epoch Time	123s	264s	97s

Table 2: *Numerical results for removing bottleneck layers*

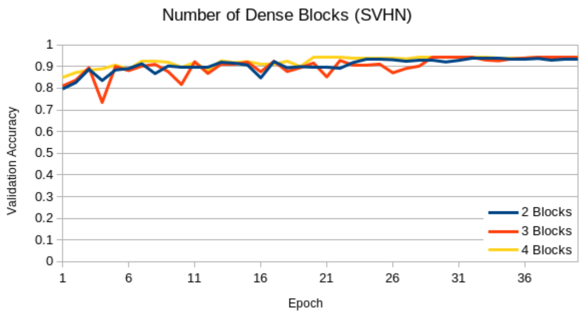

Figure 7: *Performance of varying number of dense blocks on the SVHN dataset*

and ideally should be treated as hyper-parameter based on the dataset, something that the paper did not discuss. There is an obvious limitation to the number of dense blocks that the network can have based on the dimensions of the input however, since the transition blocks between the dense blocks are reducing the dimensions and this is why we were not able to test SVHN with 6 dense blocks. For this study we used a batch size of 100 for achieving fast epoch speeds, and memory capacities of the graphics card used was a limitation that did not allow testing with a single dense block on either datasets.

i.7 Varying Growth Rate Within Blocks

In the original implementation, the growth rate k, i.e. the number of filters in each 3x3 layer, is fixed throughout the network. We decide to see what would be the effect of changing the growth rate within a block. It might seem that features computed towards the later layers would be more complex than earlier features and we might need less of them. For completeness, we try both having more filters in the beginning and in the end.

To do this, we introduce what we called the *skew factor* or simply *skew*. A positive skew means more convolution filters towards the last layers, a negative skew means more convolution filters towards the first layers. We make sure that regardless of its value the total amount of filters per block remains the same and that the number of filters per bottleneck layer unchanged.

If k is the original number of filters per convolution layer and s is the skew factor and the difference between a layer and the middle of the block is i (could be positive or negative), then that layer would have $k + si$ 3x3 convolution filters. We round i away from zero when the number of layers per block is even so that the total amount of filters stays the same. For example, with 6 layers, $k = 12$ and skew factor $s = -2$, we get the following number of filters (starting with the earlier layers): 18, 16, 14, 10, 8, 6. For $s = 3$ we get: 3, 6, 9, 15, 18, 21. Note that for $s = 0$ we get the original network.

For this ablation we train a 40-layer DenseNet with 3 dense blocks and 6 3x3 layers per block. We train it on augmented CIFAR-10. The results are shown in **Figure 8**.

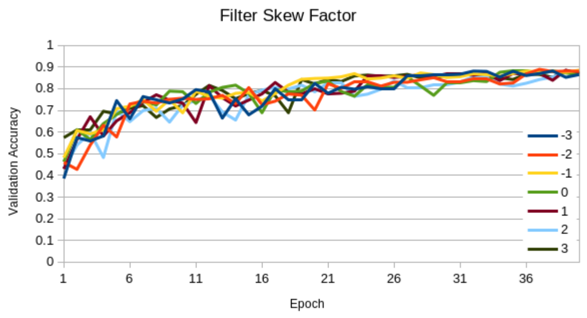

Figure 8: *Performance of varying skew factor within blocks*

It seems that changing the skew factor did not impact the results, or at least not in a way that is noticeable when training for only 40 epochs. This is quite impressive since some of these instances had a disproportionate amount of the filters either at the beginning or at the end of dense blocks. A possible reason for this is that since all data is preserved through the layers, it does not matter as much where most of the computation is done. It would be worth exploring how extreme can we make the difference between earlier and later layers before the network does not perform as well.

ii. Hyper-Parameter Experiments

ii.1 Growth Rate Coefficient in Bottleneck Layers

The original implementation of this network uses a specific value of $4k$ bottleneck layer features, where k is the global growth rate in the network. We were not able to find an explanation of why the arbitrary value 4 was picked for this and hence decided that it should be further studied. We test the values 2, 4, 6, and 8 as the growth rate coefficient of bottleneck layers and observe the change in test performance. We did expect higher values to give better accuracy and worse computational efficiency but decided to study the trade-off. The results of this study on CIFAR-10 is shown below in **Figure 9**.

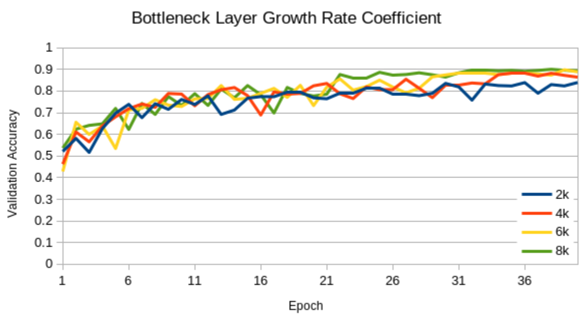

Figure 9: *Performance of varying growth rate coefficients in bottleneck layers*

As expected, we observe that higher coefficients give better results earlier in the training process. However, for further analysis we need to consider some numerical results, shown in **Table 3**

With every increase we are getting about a 100,000 extra parameters in the model. As visible in the graph on **Figure 4**, we achieve 87.65% test accuracy with 8k filters on epoch 22 which gives it a training time 3520s to achieve that accuracy. However, we achieve the same test accuracy with 4k after 39 epochs, which gives a training time of 4797s to achieve that accuracy. This shows that although the training epochs are faster with less features, it takes less overall time to reach higher test accuracy with more features and so we observe that less features are not *truly* more efficient if the same accuracy is desired. Although it might be more efficient to use less features on the bottleneck layers due to graphics card capacities that cause a bottleneck of the size of matrices the computation can have, we observe that with enough resources, a limited time-frame would favor having more features through this hidden hyper-parameter. Observing our study, we assume that the paper's arbitrary choice of the value $4k$ acts as a middle-ground of this trade-off and we would recommend treating this as a hyper-parameter while using this network for different datasets.

iii. Performance Analysis Experiments

iii.1 Evaluating Claims of Regularization

In two specific sections of the paper, the original researchers claim that there are implicit effects of regularization that stem from the underlying architecture. They claimed that the densely connected layers as well as the bottleneck layers provide regularization. The paper suggests that these were speculations based merely on observations so we decided to explore to what extent this regularization was present.

We evaluated two conditions for this experiment: size of data and the presence of the bottleneck layer. We wanted to test with various sizes of datasets in order to see if DenseNet could regularize on smaller datasets. It's quite common that complex CNNs overfit on small datasets. To test this we broke the datasets into sizes of 100, 1000 and the original 50000. For each of the dataset size we also turned toggled the bottleneck. This resulted in six different runs and the results are highlighted in **Figure ?? - Figure ??**.

An interesting observation is that overfitting seems to occur when the bottleneck is in place, contrary to the claims made in the paper. This seems to be inconsistent with the fact that the aim of the bottleneck is to reduce the number of weights being trained on. It could be that, even though the bottleneck reduces the number of paramaters being operated on, it still adds to the complexity of the

	2k	4k	6k	8k
Accuracy	83.96%	88.34%	89.85%	90.06%
Error	16.04%	11.66%	10.15%	9.94%
Params	154,402	257,218	360,034	462,850
Epoch Time	86s	123s	142s	160s

Table 3: *Numerical results of varying bottleneck layer growth rate coefficient*

model by making it deeper. The effects of overfitting are quite apparent in the original dataset when the bottleneck layer is not included. However, these observations should perhaps be taken with a grain of salt given that the original experiment is performed with 100+ epochs where as ours only went to 40.

V. Conclusion

We started our study by observing instances of changes that we believed could expose the structure of DenseNet the most. Our study was composed of three main categories; changes to architecture, hyper-parameter tuning, and performance study based on the claims made in the paper. We mainly focused on architectural changes as these seemed more interesting than hyper-parameter tuning. Through our architectural changes we have discovered some hidden hyper-parameters that were not reported in the paper, such as the number of dense blocks in the network. We have also created new hyper-parameters under our study involving change in growth rates of individual blocks. Due to our computational limitations, we were not able to explore the network with more depth which as reported by the authors was the record-breaking version of DenseNet. We believe a potential continuation from where we left off could involve attempts to perform the studies we have reported but on deeper versions of DenseNet, as well as testing the hyper-parameters that were reported by the authors to be accurate versions that provide the best performance. DenseNet is a fascinating yet simple addition to the family of convolutional neural networks and we believe the structure is extremely robust so various kinds of datasets and hence deserves a lot of attention in the ablation field for newer models to inspire from.

VI. Statement of Contributions

Changes to number of dense blocks, bottleneck layer growth rate factor, and dropout layer were tested by Nima Adibpour. Re-ordering of composite functions, average vs. max pooling, FashionMNIST, and regularizing effects of bottleneck layer were tested by James Brace. *n*-Previous layers, varying growth rate within blocks, effects of bottleneck layer on smaller depths were tested by Paul-Andre Henegar. The report was written as a mutual effort between all authors. The graphs were generated by Nima Adibpour. We have also used Google Computation Cloud resources [4] (4 CPU, 15GB RAM, NVidia K80) for performing all studies mentioned in this report.
We hereby state that all the work presented in this report is that of the authors.

References

[1] DenseNet. *Densely Connected Convolutional Networks* arXiv:1608.06993 [cs.CV]
PDF: `https://arxiv.org/pdf/1608.06993v5.pdf`.

[2] Keras
The Python Deep Learning library

[3] Dense Net in Keras
GitHub Repository by titu1994 with a Keras 2 based implementation of DenseNet

[4] Google Cloud
Google Cloud Computing, Hosting Services, & APIs

[5] Average Pooling. *Network In Network* arXiv:1312.4400 [cs.CV]
PDF: `https://arxiv.org/pdf/1312.4400.pdf`.

VII. APPENDIX

i. Evaluating Performance on Fashion MNIST

Given that the original report focused on very complex datasets, we though it would be fitting to analyze how well this architecture performs on simpler datasets. We ran the model on the dataset using the same setup as the original DenseNet targeted at CIFAR-10, including the use of the bottleneck layer. We predicted that a complex model like DenseNet would tend to overfit a simpler dataset such as Fashion MNIST. However, as seen in **Figure 10**, this is not the case. In fact, this model performs spectacularly well on Fashion MNIST, which not only shows the robustness of this model but also exhibits the regularization effects of the underlying architecture.

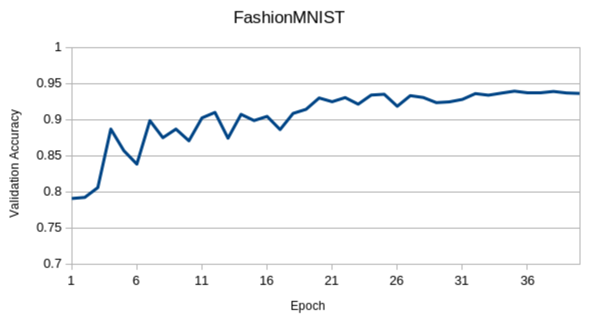

Figure 10: *Performance of DenseNet on Fashion MNIST*

ii. Extra Figures

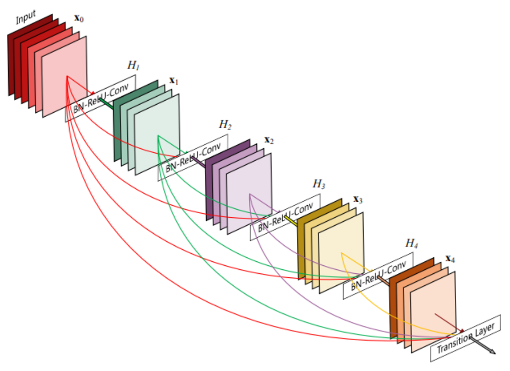

Figure 11: *The core architecture of Densely Connected Convolutional Networks [1]*

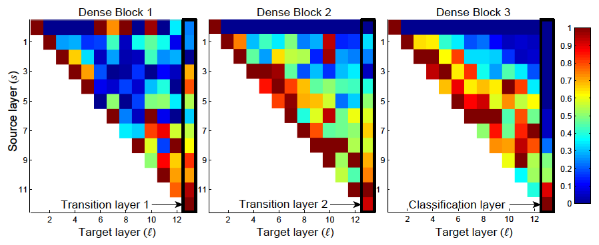

Figure 12: *The average absolute filter weights of convolutional layers in a trained DenseNet.*

iii. Executive Summary

The purpose of this report was to understand one of the latest state-of-the-art Convolutional Neural Networks, DenseNet, to explore the robustness of Densely Connected Convolutional Neural Networks. The idea behind DenseNet is to capitalize on key observations made on ResNet and Highway Networks: that creating connections from early to later layers improve performance by persisting gradient information as the networks gets deeper. This resulted in profound performance similar to the latest state-of-the-art, but with much fewer parameters. We performed all our experiments, except for one, on an augmented CIFAR-10 dataset.

In performing our ablation, we looked to make changes to the underlying architecture, its hyperparameters, and in general observe the effects in doing so. We started off by only using n-connections, where $n < N$, to see if connections become redundant as the networks becomes deeper. This resulted in no observed improvement, but offered interesting insights about the effects that connections have on regularization. Next, we added additional dropout layers to the models, this resulted in decreased validation performance since it probably counteracts the point of propagating data to deeper layers. As a quick experiment, we changed all the average pooling layers for max pooling layers. This was done in part because we did not understand the implications of average pooling. What we observed was that max pooling performed slightly better than average pooling when ran with 40 epochs. One of our largest experiments was to try all possible combinations of composite functions. The original composite function goes as following: Batch Normalization ReLU Activation Bottleneck 3x3 Convolutional Layer. When performing the various combinations, we coupled the bottleneck and 3x3 Conv Layers which resulted in a total of 6 unique combinations. INSERT RESULTS HERE. We also wanted to measure the effect bottleneck layers had on performance. It was discovered even though bottleneck layers reduce the number of global parameters, the models with the bottlenecks actually train faster. Naturally, we experimented with different numbers of dense blocks since it is the core of the architecture. We observed that the ideal number of dense blocks is between two and four given our experiments on CIFAR-10 and SVHN with 40 epochs. More dense blocks led to overfitting, which is expected given that we are adding to the complexity of the model. Also, this optimal range of dense blocks also led to faster convergence. This was observed in both CIFAR-10+ and SVHN. This led us to conclude that the number of dense blocks can be treated as a hyper-parameter based on the dataset being used. In addition to the number of dense blocks present, we also played around with varying growth rates per block. In the traditional architecture the growth rate is the same per block. However, given the the fact that blocks later in the model have a greater effect on the classification layer, we sought to see what happens if we have a greater growth rate in the earlier blocks than in the later blocks. We were surprised to discover that differing the growth rate per block resulted in the same performance. Next we did some hyperparameter-tuning by changing the growth rate coefficient in bottleneck layers. In the original architecture the growth rate coefficient was 4, which resulted in a growth rate of 4k. We tried 2, 4, 6, and 8 as our options. Higher coefficients gave better accuracy by a small margin but resulted in much larger training times. We concluded that 4 was chosen due to a weighted training time/ accuracy trade-off. Lastly, we evaluated the claims of regularization made in the paper as well as evaluated the performance on fashion MNIST. For regularization measurements we tried various sizes of datasets with the option of bottleneck being turned off. The outcome was that, contrary to the claims made in the paper, bottleneck hindered regularization and led to more overfitting than with it being included. We then ran the original architecture on Fashion MNIST and were surprised to see that it exhibited amazing performance and showed no signs of overfitting. This displayed the robustness of the model on less complex datasets, whereas other complex models such as ResNet would have perhaps overfitted.

Figure 13: *Motivation for this study*