Reproducibility report of Densely Connected Convolutional Networks

Zhengxin Chen
email:zhengxin.chen@mail.mcgill.ca

Evelyn Cao
email: tianyu.cao@mail.mcgill.ca

Amelia Cui
email: wen.cui@mail.mcgill.ca

1 Introduction

Convolutional neural networks (CNNs) have become a trivial method in machine learning in the last few years. Many structures were proposed to make more powerful CNN, but most of them focused mainly on increasing the depth (ResNet) or width (Inception of GoogleNet) of the network. In this paper[1] the authors proposed a novel architecture—densely connected convolutional network (DenseNet), which instead focused on making full use of the feature-maps to improve the performance. The authors claimed that their model achieved state-of-the-art performance with less parameters and computation.

In this work, we aim to reproduce their findings, verify their claims, and perform additional experimental results to provide further investigation into the model. Precisely, we test the performance of DenseNet on CIFAR , analyse the impact of several hyper-parameters and explore methods of improving memory efficiency.

2 Scope of reproducibility

As CNN becomes increasingly deep, the problem of gradient vanishing and information loss occur. The paper addressed this problem by a simple connectivity pattern: connect all layers (with matching feature-map sizes) directly with each other to ensure maximum information flow between layers in the network. Figure 1 shows the layout of a single dense block. Figure 2 shows the architecture of a 3-dense-block DenseNet. In this reproducibility study, our main goal is to verify the following claims of the original paper:

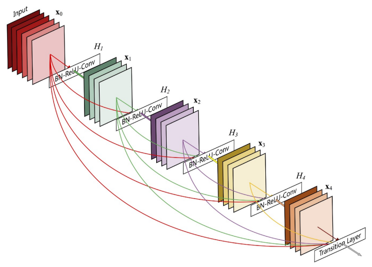


Figure 1: A 5-layer dense block with a growth rate of k = 4.

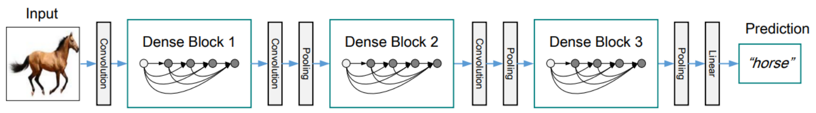


Figure 2: A deep DenseNet with three dense blocks.

- **Accuracy** DenseNet tends to achieve better performance with increasing growth rate and depth.
- **Deep supervision** DenseNet has the advantage of advantage of improved flow of information and gradients throughout the network. This helps training of deeper networks.

- **DenseNet-BC** The Bottleneck - Compressed DenseNets offer further performance benefits, such as reduced number of parameters, with similar or better performance

The remainder of this work is structured as follows. In Section 3, we introduce the model proposed in the original paper, including the dataset, hyper-parameter setting and computation requirements. Section 4 presents the replicated results and compares them to the original paper.

3 Methodology

3.1 Model descriptions

Dense Convolutional Network connects each layer to every other layer in a feed-forward fashion. Whereas traditional convolutional networks with L layers have L connections, one between each layer and its subsequent layer, DenseNet has $L(L+1)/2$ direct connections. For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. DenseNet-BC is an improved version of DenseNet. It contains an additional bottleneck layer (1*1 convolution) before each convolution layer to reduce the number of input feature-maps of dense blocks. Moreover, it does a compression in transition layers by a factor of $0 < \theta < 1$ to further improve model compactness.

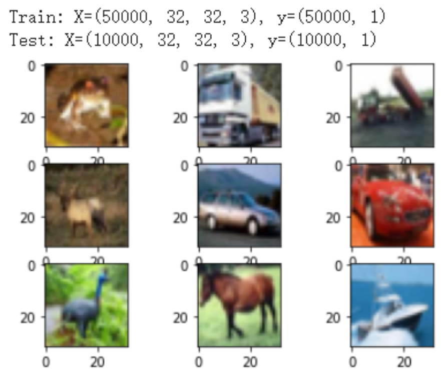
Basically, DenseNet has two characteristic hyperparameters, growth rate and number of layers in each dense block. First, we re-implement three basic DenseNet structure each with three dense blocks, their configurations are "L = 40, k = 12", "L = 100, k = 12" and "L = 100, k = 24". Second, we re-implement four DenseNet-BC structure each with four dense blocks, table 1 shows the exact configurations.

Model	Growth rate	Dense-block layer
DenseNet121	32	[6, 12, 24, 16]
DenseNet161	48	[6, 12, 36, 24]
DenseNet169	32	[6, 12, 32, 32]
DenseNet201	32	[6, 12, 48, 32]

Table 1: DenseNet-BC configurations

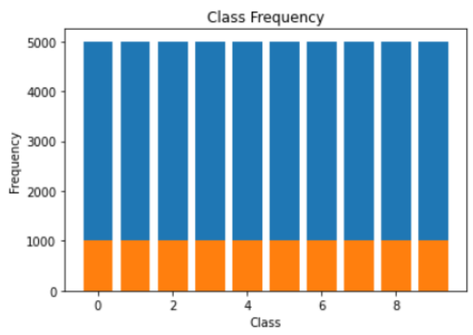
3.2 Datasets

The paper tested the DenseNet models on three benchmark datasets (CIFAR, SVHN, and ImageNet). Since they are doing similar classification tasks and the limit of hardware, we only focus on CIFAR-10. The CIFAR-10 dataset consist of colored natural images with 32×32 pixels drawn from 10 classes. We split it into 50000 training data and 10000 test data, and did normalization. Figure 3 shows the sample images and class distribution.



```
Train: X=(50000, 32, 32, 3), y=(50000, 1)
Test: X=(10000, 32, 32, 3), y=(10000, 1)
```

(a) Sample



(b) Class Distribution

Figure 3: CIFAR-10 data visualization

3.3 Hyperparameters

In order to match the original experiments as closely as possible, we tried to use the same hyperparameters as the authors. But we did not run the same epoch because it took too much time. If the required hyperparameters for the experiments were not mentioned in the original paper, we relied on the default parameters given in the configuration files of the original implementation.

3.4 Experimental setup and code

We do not use the code from the authors because they used the Lua language. We instead use a python vision code from GitHub and GitHub with some minor modification. We re-implement some experiments based on the description provided in the paper. Furthermore, we improve and extend upon the work of DenseNet by providing additional experiments and results. We reproduce the experiments on Colab with T4 and P100 GPU

3.5 Computational requirements

We ran experiments on P100 GPU.

Time to train each model:
DenseNet(L=40,k=12): about 47sec per epoch with batch size 64
DenseNet(L=100,k=12): about 4min per epoch with batch size 64
DenseNet(L=100,k=24): about 15min per epoch with batch size 64
DenseNet121:about 3min per epoch with batch size 128
DenseNet161:about 3min-20sec per epoch with batch size 128
DenseNet169: about 4min per epoch with batch size 128
DenseNet201:about 3min per epoch with batch size 128

When we try to use high growth rate and larger block layers DenseNet models, we need GPU with at least 18 Gib memory size.

4 Results

In general, DenseNet-BC architecture has the best performance. With the number of epoch increasing, every model would achieve higher accuracy. Moreover, the deeper models with higher growth rate and larger block size tend to have better performance without overfitting.

4.1 Results reproducing original paper

In summary, we run experiments on different number of epochs to test the relationship between number of epochs and accuracy. We also perform experiments on different growth rate, block layers for both the basic DenseNet and DenseNet-BC to test the effect of hyper-parameters on this model. The details are shown in the following sections.

4.1.1 Results of basic DenseNet

We run each basic DenseNet for just 20 epoch instead of 300 in the original paper considering limited running time, but it is sufficient to see the impact of growing growth rate and depth. We find that the Densenet with L=100 and k=24 achieved the highest accuracy of 86.68%. We conclude that as the growth rate or depth increasing, the model is more expressive and thus achieve better performance. Note that we do not find the minor overfitting problem mentioned in the original paper when k=24, we think that's because our training epoch is low. Table 2 contains the detailed results.

Model	Depth	Accuracy	Params
DenseNet(k=12)	40	83.5%	1.06M
DenseNet(k=12)	100	86.13%	7.08M
DenseNet(k=24)	100	86.68%	28.07M

Table 2: Test results of basic DenseNets. k denotes growth rate.

4.1.2 Results of DenseNet-BC

The original paper tested them on ImageNet, but we test on CIFAR-10 because ImageNet requires more than our accessible GPU memory and too much running time. First, analyse the impact of training epoch and find that the model performance tend to improve as the number of epoch increases. Figure 4 shows the learning curve of DenseNet121 model.

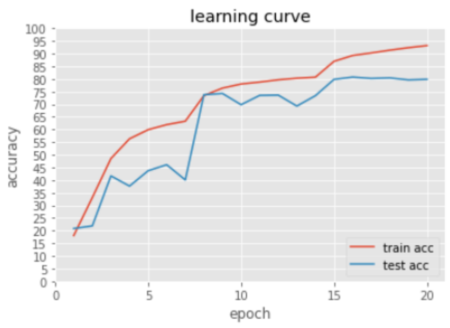
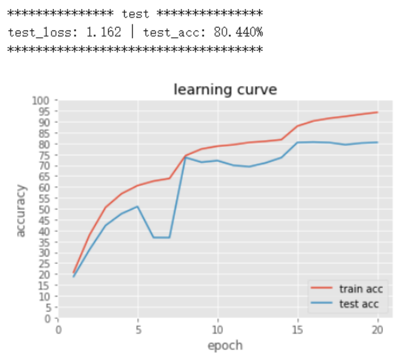


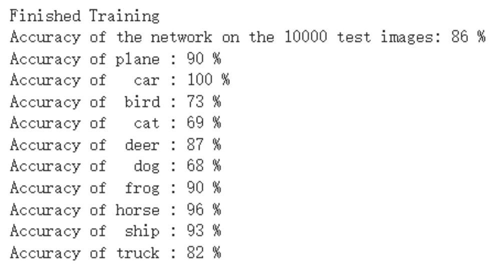
Figure 4: DenseNet121 leaning curve

Second, we run the four DenseNet-BC models each for 20 epoch. We find that the model tends to have better performance when the number of parameters increases, and DenseNet201 achieves the highest test accuracy 88%. Moreover, among the ten classes, all models tends perform better on car and plane, but relatively bad on cat and bird.



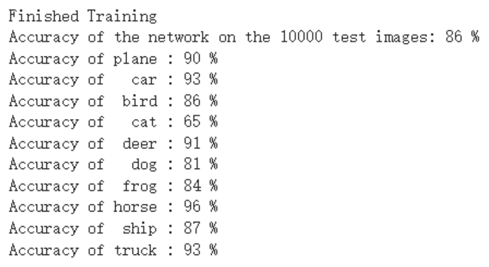
```
*************** test ***************
test_loss: 1.162 | test_acc: 80.440%
************************************
```

(a) DenseNet121



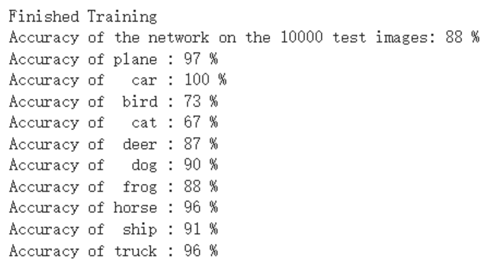
```
Finished Training
Accuracy of the network on the 10000 test images: 86 %
Accuracy of plane : 90 %
Accuracy of   car : 100 %
Accuracy of  bird : 73 %
Accuracy of   cat : 69 %
Accuracy of  deer : 87 %
Accuracy of   dog : 68 %
Accuracy of  frog : 90 %
Accuracy of horse : 96 %
Accuracy of  ship : 93 %
Accuracy of truck : 82 %
```

(b) DenseNet161



```
Finished Training
Accuracy of the network on the 10000 test images: 86 %
Accuracy of plane : 90 %
Accuracy of   car : 93 %
Accuracy of  bird : 86 %
Accuracy of   cat : 65 %
Accuracy of  deer : 91 %
Accuracy of   dog : 81 %
Accuracy of  frog : 84 %
Accuracy of horse : 96 %
Accuracy of  ship : 87 %
Accuracy of truck : 93 %
```

(c) DenseNet169



```
Finished Training
Accuracy of the network on the 10000 test images: 88 %
Accuracy of plane : 97 %
Accuracy of   car : 100 %
Accuracy of  bird : 73 %
Accuracy of   cat : 67 %
Accuracy of  deer : 87 %
Accuracy of   dog : 90 %
Accuracy of  frog : 88 %
Accuracy of horse : 96 %
Accuracy of  ship : 91 %
Accuracy of truck : 96 %
```

(d) DenseNet201

Figure 5: Results of DenseNet-BC

Third, we compare the performance and parameter efficiency of DenseNet-BC with basic DenseNet. As figure 6 illustrates, DenseNet121 uses more parameters but gets lower accuracy than DenseNet(L=40,k=12), which somehow conflicts the claim of the paper. We think that's because our training epoch is small, so the models are not fully trained to show their real ability. On the other hands, DenseNet-BC achieves slightly higher best accuracy using significantly less parameter (12.3M vs 28.08M). Therefore we conclude that generally DenseNet-BC shows higher parameter efficiency with similar or better performance.

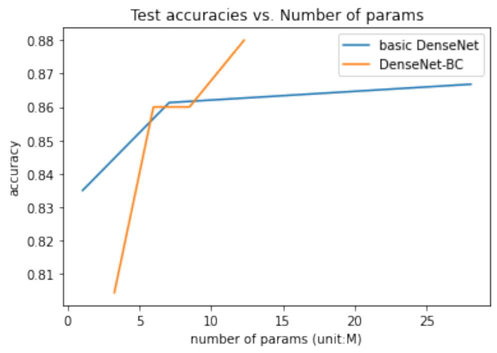


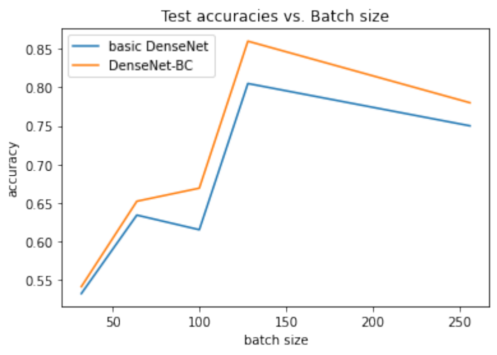
Figure 6: Compare parameter efficiency

4.2 Results beyond original paper

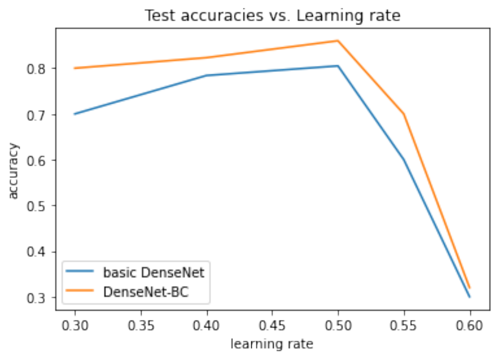
Beyond the experiments in the original paper, we further investigate the impact of more hyper-parameters (batch size and learning rate), and explore methods to improve memory efficiency.

4.2.1 Impact of more hyper-parameters

Running for the same epoch, we intend to find the impact of different batch size and learning rate on DenseNet. Figure 7 illustrates that batch size 128 gives the best performance for both the basic DenseNet and DenseNet-BC, and DenseNet-BC gives better performance than the basic one in all cases, which is align with our finding above. From the plot on the right, the most appropriate learning rate for our models is 0.5 as too large learning rate will over-shoot the local optimum.



(a) acc vs. bs



(b) acc vs. lr

Figure 7: Acc vs batch size and learning rate

4.2.2 Methods to improve memory efficiency

The implementations of DenseNet we used tend to be memory-hungry, so we further investigate and find the paper [2] and a implementation from GitHub. According to the paper, the number of intermediate feature maps generated by batch normalization and concatenation operations grows quadratically with network depth. The implementation uses checkpointing to compute the Batch Norm and concatenation feature maps. These intermediate feature maps are discarded during the forward pass and recomputed for the backward pass.

Using this efficient implementation the running time for the same three basic DenseNet become about 21sec/epoch, 2.3min/epoch and 5.6min/epoch, which are significantly faster.

5 Discussion

The original paper [1] proposed a densely connected neural network which introduced direct connections between any two layers with the same feature-map size. Throughout this work, we have conducted several experiments to reproduce the main results from the research of DenseNet. The results of our reproducibility study provide support for some of their claims. Specifically, our experiments show that the test accuracy of DenseNet on CIFAR-10 can be improved by increasing the growth rate or depth. DenseNet-BC using additional bottleneck layer and compression in the transition layer shows the similar trend, the test accuracy keeps improving as the parameter of the model grows. Furthermore, DenseNet-BC can achieve significantly better parameter efficiency than the basic DenseNet. All the results support the claim in the original paper that DenseNets tend to yield consistent improvement in accuracy with growing number of parameters and is characterised by high parameter efficiency.

Nonetheless, limited by hardware, our models were not fully trained with enough epoch and we did not use all the datasets used in the original paper. So we can reproduce similar trends, but not the exact accuracy as the original paper.

5.1 What was easy, what was difficult

The implementation details and hyperparameters are clear in the original paper. Beyond that, since the DenseNet publicly accessible and well structured implementations on the Internet. As such, getting started with the experiments is straightforward.

Nonetheless, the original code is written by Lua language which we cannot use, and the other implementations are not completely the same with each other, it's tricky to choose one implementation that we can run on our computer and work similarly to the original paper. Moreover, reproducing the original results turned out to be far from trivial as the setup of the experiments required some modifications to the code. Lastly, The models and datasets used in the paper are way more huge than what our hardware can compute, so getting the baseline model to work and obtaining numbers similar to those reported in the respective papers was challenging.

6 Statement of Contributions

Everyone contributes to conducting the experiments and writing the write-up.

References

[1] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. *arXiv:1608.06993*, 2016.

[2] G. Pleiss, D. Chen, G. Huang, T. Li, L. van der Maaten, and K. Q. Weinberger. Memory-efficient implementation of densenets. *arXiv preprint arXiv:1707.06990*, 2017.