Densely Connected Network (DenseNet) for image classification

Clément Gousseau - Stefan Gramfält - Niklas Qvaforth

DD2424 Project report

Abstract. The problems of vanishing gradients and similar problems have raised when the neural networks started to become deeper and deeper. To solve this problem scientists have tried different techniques such as stochastic depth, identity connections and similar. DenseNet is a new approach that make use of the fact that some of the layers are redundant or do not contribute so much to the result. This is done by concatenating the results in between layers instead of using the sum, allowing for the reuse of parameters. This have given promising results and has high performance compared to regular convolutional networks. This report examines different settings of DenseNet to evaluate its performance regarding accuracy and parameters used to regular convolutional networks. Results show that DenseNet is indeed performing well compared to other networks, with less parameters and high accuracy.

1 Introduction

Convolutional neural networks have become dominant for visual object recognition. New and faster hardware has made the progress go forward allowing the networks to be more computational expensive. This makes it possible to have deeper networks which has proved to be more successful. But with this progress new problems have raised. As the gradient has to pass more and more layers the risk of it getting lost along the way and problems related to that has increased. This is called the vanishing gradient problem and is caused by the gradients being products of weights that can be very small, causing them to eventually vanish. This problem and similar is something recent publications try to address and find a solution to. ResNets and Highway Networks try to bypass signals by using identity connections[1]. FractalNets combine several parallel layers with convolutional blocks and maintain several short paths [1]. ResNet use a technique called "stochastic depth" which is as a way of reducing the problem by randomly dropping layers during training allowing better gradient flow [1]. This idea of stochastic depths and the result that it makes no big difference when some layers are dropped during training, gives an indication that some layers are redundant or contribute very little to the result[1]. Even though ResNet uses stochastic depth it still needs a lot of parameters because of the property that all layers have its own weights. DenseNet solves this problem and preserver the stochastic property with the idea of having all layers connected with each other,

each layer receives additional information inputs from all preceding layers. Because DenseNet instead of summation which is used in between layers in ResNet, is using a concatenation of parameters, less parameters is needed to train the network it also decreases vanishing gradients problem and has a regularizing effect [1]. DenseNet seems to be very promising with a lot of positive effects and not so many identified downsides. In this report we are using an implementation of DenseNet to evaluate how it performs with different settings. This is then compared to and evaluated to known results of other convolutional neural networks. We also try to find possible downsides of DenseNet.

2 Background

2.1 Neural networks

The human brain consists of 100 billion nerve cells or *neurons*, these communicate with electrical signals that are short lived impulses in the voltage of the cell wall membrane [2]. The inter-neuron connections are mediated by electrochemical junctions called synapses, these are located on branches of the cell called dendrites. Each neuron receives thousands of connections from other neurons and are constantly receiving a multitude of incoming signals. These signals does eventually reach the cell body, in the cell body the signals are integrated or summed in some way, and if the resulting signal exceeds some threshold the neuron will generate an voltage impulse in response to other neurons [2]. This is transmitted via a branching fiber called an axon. When determining weather an impulse should be produced or not, some incoming signals produce an inhibitory effect and tend to prevent generation of the impulse, while others are excitatory and promote impulse generation[2].

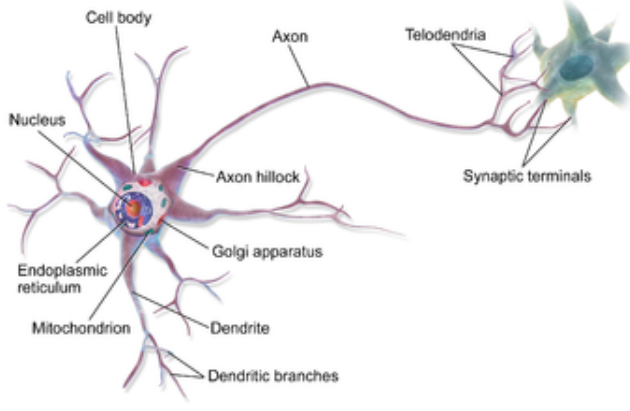

Fig. 1. Visualisation of biological network

The architecture and style of the brain are used and incorporated when designing an artificial neural network [2]. When creating a artificial neural network the neurons become nodes in our system. The synapses are modeled by a single number or weight so that every input is multiplied by this weight before forwarded to to the "cell body". In the "body" the signals are summed up together by simple arithmetics, like addition to supply node activation. Output is produced depending if it exceeds a threshold or not [2]. The term network is used for any type of system that use artificial neurons, from a single node to a large collection of nodes. In real neurons the strength of the synapse signal may be modified so that the behavior of each neuron can change or adapt to its input stimulus [2]. In an artificial environment this is equivalent to the altering of the weights values [2]. In this way the neural network can adapt to certain patterns and match a target [2]. This process as a whole including pattern presentation, criteria for termination and so on sums up and forms the training algorithm. The result should be ready to take on unknown data, aka the network is *generalizing well* [2].

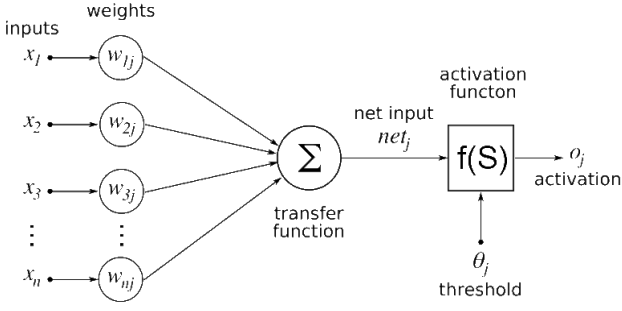

Fig. 2. Artificial neural network

2.2 CNN, Convolutional neural networks

Convolutional Neural Networks "CNN", are a specialized kind of neural network (ANN) for processing data that has a known grid-like topology [3]. CNN's were one of the first working deep networks trained with back-propagation [3]. General back propagation were earlier considered to have failed and why CNN's were a success is unclear. A convolutional network are simply neural networks that instead of a matrix multiplication uses a convolution instead of matrix multiplication[3].
The convolution between two functions $f(x * w)$ can be described of the integral [3]:

$$s(t) = f(x * w)(t) = \int_0^t x(a)w(t-a)da$$

This function is used to give a smoothed estimate of the measurement we want to obtain. In convolutional neural networks the **x** usually refers to the *input* and the **w** to the *kernel* the output is sometimes referred to as the *feature map*. when working with data on a computer the time is discretized and the discrete convolution function can be written as:

$$s(t) = f(x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a).$$

The convolution has the commutative property of which arises when flipping the kernel relative to the input, this property is usually good for writing proofs but not so important for implementation. Many neural networks implements instead a function called **cross-correlation**, which is the same as convolution but without flipping the kernel.

$$s(i,j) = f(I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n).$$

Any neural network algorithm that works with matrix multiplication should work with convolution. The main advantage of CNN's is their accuracy in image recognition problems, the drawbacks are mainly, slow training, computational expensive and the need of a lot of training data (parameters). Also the CNN's needs supervised training, at least on a high level.

2.3 Presentation of DenseNet

The idea of DenseNet. In 2015, *Microsoft Research Asia* won ILSVRC with a 3.6% top5-error. Their network, *ResNet*, is a residual learning network. The idea is to use shortcuts over layers. *ResNet* only skips over one layer. The motivation for skipping over layers is to avoid the problem of vanishing gradients.

Without residual learning one try to adjust the weights in order to fit the distribution *H(x)*.

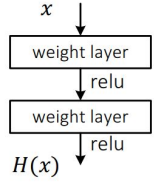

Fig. 3. Without residual learning

With residual learning one try to adjust the weights in order to fit $F(x)=H(x)-x$. Then x i added to $F(x)$ to find $H(x)$. This makes the learning easier. This principle of shortcuts is applied to every layer.

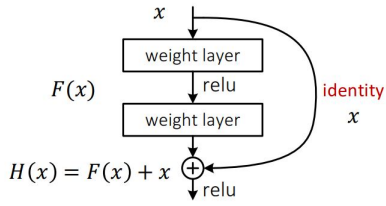

Fig. 4. with residual learning

The idea with DenseNet is not only to jump over every layer but to create a shortcut between each pair of layer

Structure of DenseNet. Here is presented an example of DenseNet structure with a number of layers L=40 and a growth-rate k=12.The growth rate is the number of features that each convolution creates. This network can be used with CIFAR-10 datasets, which is composed of colored 32x32 pixels images. Each of its images belongs to one of the 10 classes.

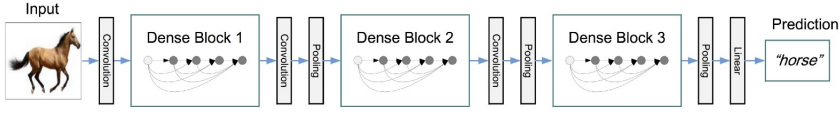

Fig. 5. General structure of DenseNet

There are four different kinds of layers

Pre-processing layer. First the input goes through a 3x3 convolution layer with stride 1 that creates 16 output channels.

Transition layers. Each of these blocks includes

- a Batch Normalization
- a ReLu activation
- a 1x1 convolution
- an 2x2 average pooling

There are two transition layers which are placed between the three dense blocks. The goal is to reduce the dimensionality of the data (this is done by the average pooling layer).

Dense Blocks. There are three dense blocks. Each of them is composed of 12 convolutions that include

- a Batch Normalization
- a ReLu activation
- a 3x3 convolution that generates k channels. The stride is 1.
- a concatenation of the input and the output of the convolution

The size of the features map are respectively 32x32, 16x16 and 8x8 for the three dense blocks. The goal of these dense blocks is to increase the number of features.

Classification layer. The final layer is composed of a Batch Normalization, a ReLu activation, a global average pooling and a 10D fully connected layer.

Advantages of DenseNet.

The vanishing-gradient problem The trend in deep learning is to create deeper and deeper networks, from 8-layers AlexNet to 152-layers ResNet. It has enabled very good results in image recognition. Now the best algorithms reach human capacities with a classification error around 3%.

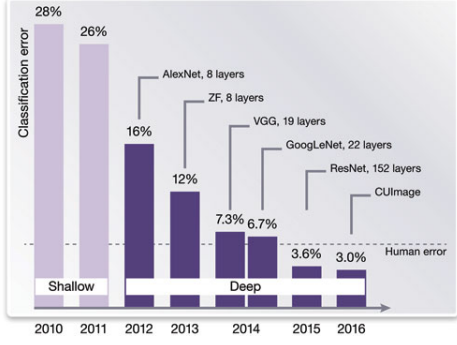

Fig. 6. ImageNet Large Scale Visual Recognition Challenge Results

But with this revolution of depth a new problem occurred. As information from the input or the gradient passes through many layers, it can vanish by the time it reaches the end. DenseNet solves this problem by using a dense network that concatenates the features created at every layer. So no information is lost during the backward or forward propagation.

Number of parameters A less intuitive advantage of DenseNet it that it requires less parameters than "traditional" networks. In "traditional" networks, some information may vanish during the propagation. Since DenseNet concatenates the features no information is lost during the propagation. Therefore there is no need to learn a feature twice, which makes the network more efficient in terms of number of parameters used.

3 Approach

The goal of this project is to give a comprehensive study of DenseNet in CIFAR-10 image clasification. This consists in two main parts:

Study of the parameters of the network. The two main parameters of a DenseNet network are the growing rate k (number of new features created at each convolution) and the depth d. We trained several networks with different settings (k,d) to find out which parameter has the biggest influence on the accuracy.

Comparison with other networks. We compared the performance of DenseNet with other CNN, regarding the number of parameters and the accuracy.

3.1 Different settings

After some empirical tests to determine which networks were possible to train with a reasonable amount of time and memory it has been decided to train 9 networks which corresponds to 3 different growing rates for 3 different depths:

depth d \ growing rate k	k=8	k=16	k=32
d=10	Test 1	Test 2	Test 3
d=13	Test 4	Test 5	Test 6
d=19	Test 7	Test 8	Test 9
d=40	Test 10	Test 11	Test 12

Table 1. The different trained networks

We also trained a network with depth 40 and growing rate 12 (test 13). This is the simplest network presented in the original DenseNet paper [1]. The goal was to check the provided accuracy (0.93).

3.2 Training

For the tests 1 to 12, the training has been done on a Tesla K80 GPU. In order to do the training in a reasonable amount of time and with good performance the learning rate has been set to:

- 0.1 for the first 24 epochs
- 0.05 from the 25th epoch to 29th
- 0.01 from 30th epoch to 34th
- 0.003 from 35th epoch to 39th
- 0.001 from 40th epoch to 45th

The training is stopped after 45 epochs, since the accuracy was stable at this point.

For test 13, we wanted to obtain the best accuracy possible. We used 300 epochs for the training, with a learning rate set to 0.1 for the first 149 epochs, 0.01 for epochs 150th to 224th and 0.01 for epochs 225th to 300th.

3.3 Comparison with other networks

We compared the results of some of DenseNet networks with other CNN networks. We used the results from the paper *"Let's keep it simple, using simple architectures to outperform deeper and more complex architectures"* by Seyyed Hossein Hasanpour et al.[4]

4 Results

4.1 Accuracy on the test set for the different settings

Settings with short training (tests 1 to 12). On figure 7 one can see that the loss error decreases pretty quickly during the first epochs and then starts oscillating. After epoch 25th the learning rate is decreased, which enables a reduction of the the loss error. Then the learning rate is slowly reduced until the loss is stable.

depth d \ growing rate k	k=8	k=16	k=32
d=10	78.0	84.3	88.2
d=13	81.4	86.6	89.0
d=19	86.2	89.0	90.9
d=40	90.2	91.0	91.5

Table 2. Accuracy on the test set for the different settings

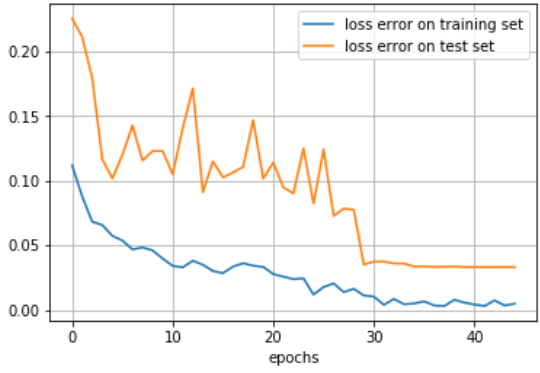

Fig. 7. evolution of the loss for a short training (depth=19/k=32)

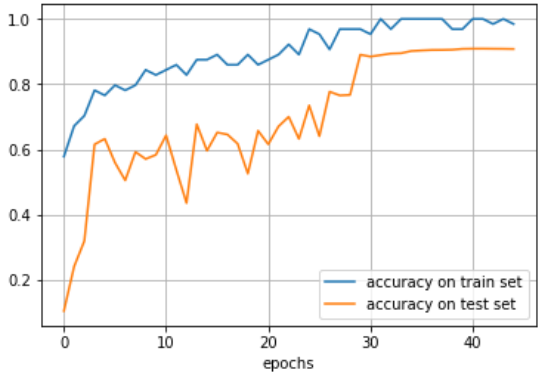

Fig. 8. evolution of the accuracy for a short training (depth=19/k=32)

Setting with long training (test 13). This test gave very good results with a final accuracy of 93.2%. Unfortunately we could not afford to repeat such trainings with different settings since Test 13 took 30 hours to complete. Therefore for the rest of the report we will focus on the tests with short trainings.

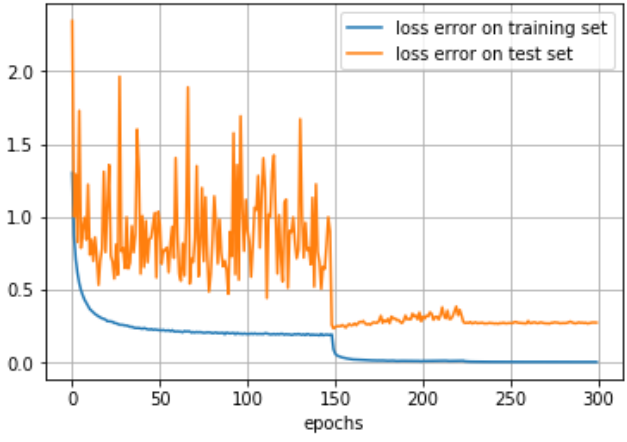

Fig. 9. evolution of the loss for a long training (depth=40/k=12)

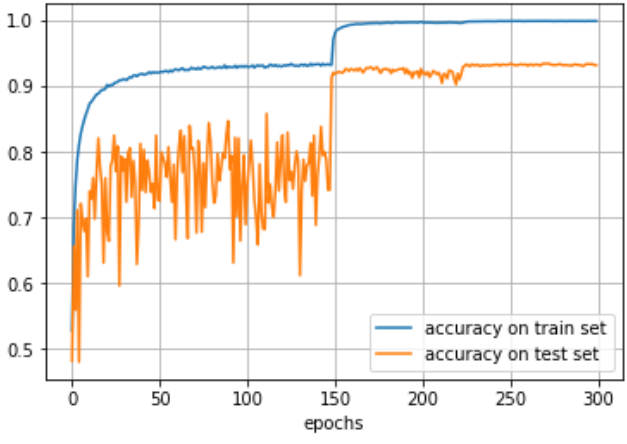

Fig. 10. evolution of the accuracy for a long training (depth=40/k=12)

4.2 Study of the performance for different settings

Influence of the depth and growing rate on the accuracy. Figure 11 and 12 show that for a given training strategy one can improve the accuracy of the network by modifying two parameters: the depth of the network and the growing rate k. Obviously the accuracy increases with the depth and the growing rate.

The interesting point is that two models with different depth and growing rate may have roughly the same accuracy. For example, the network with depth=13/k=32 and the network with depth=19/k=16 has the same accuracy (0.890). Likewise the settings depth=19/k=32 and depth=40/k=16 have similar accuracy (respectively 0.910 and 0.907)

So one can chose one or the other network depending on what performance needs to be optimized (training time, number of parameters,etc.).

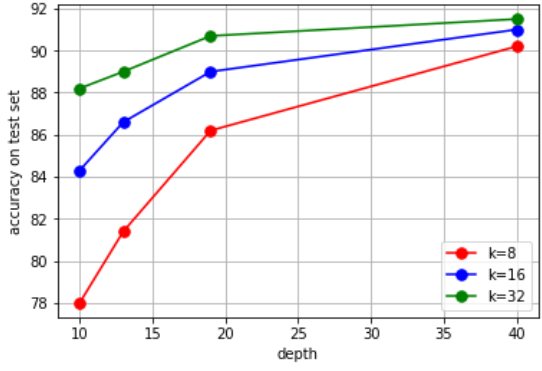

Fig. 11. Accuracy depending on the depth (results grouped by growing rates)

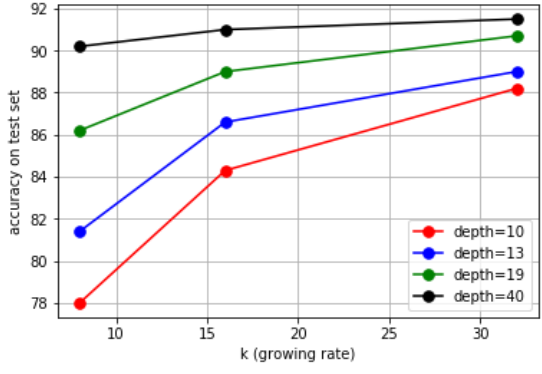

Fig. 12. Accuracy depending on the growing rate (results grouped by depths)

Influence of the depth and growing rate on the training time and accuracy. On Figure 13 one can see that for the most simple networks, a little longer training enables a much higher accuracy. For example with k=8 using a network with depth 19 instead of 10 increases the accuracy by 8%, but only increases the training time by 0.75 minute/epoch. On the contrary for more complex models, a large amount of training time is needed for a small gain in accuracy. For instance with k=32, using a network with depth 40 instead of a network with depth 19 increases the accuracy by 0.8 but multiplies the training time by 4.

In 4.2.1. we noticed that the network with depth=13/k=32 and the network with depth=19/k=16 had the same accuracy (0.89). Using the first setting would save 0.53 minute per epoch, that is to say 24 minutes for a 45 epochs training. Likewise, using a network with depth=19/k=32 instead of depth=40/k=16 would save 3.42 minute per epoch, that is to say 2h23 for a 45 epochs training.

So in order to save time one should opt for a shallow network with a large growing rate.

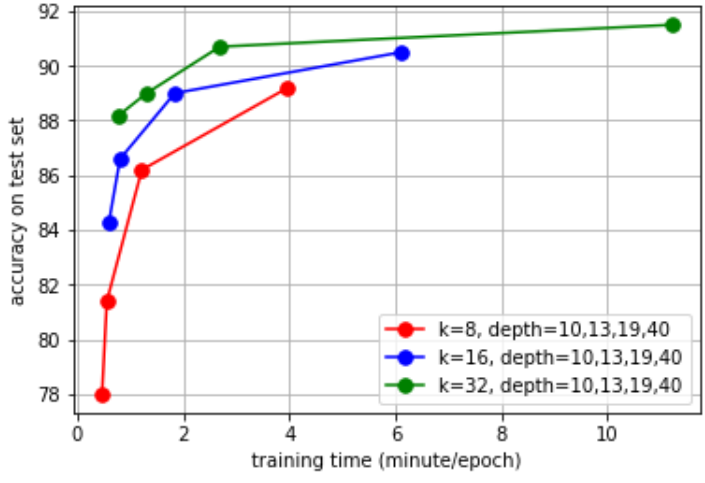

Fig. 13. Evolution of the accuracy with the training time for different settings

Number of parameters. It is not surprising to see that the accuracy increases with the number of parameters. Let us consider again the two networks depth=13/k=32 and depth=19/k=16 which have the same accuracy (0.89). The first setting uses 783.10^3 whereas the second setting uses 441.10^3 parameters. The settings depth=19/k=32 and depth=40/k=16 have roughly the same number of parameters.

So to have a more compact model one should opt for a deep network with a small growing rate.

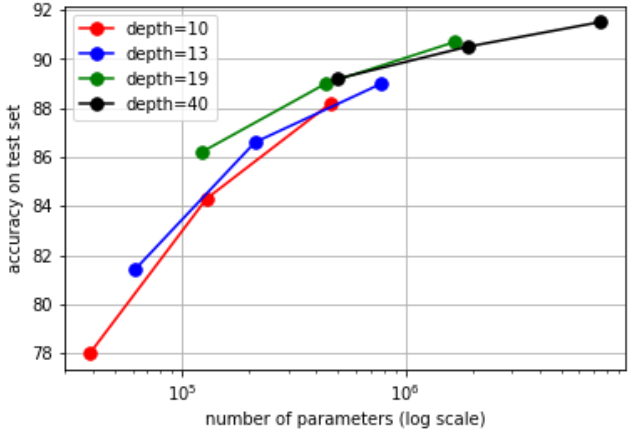

Fig. 14. Accuracy depending on the number of parameters (growing rates)

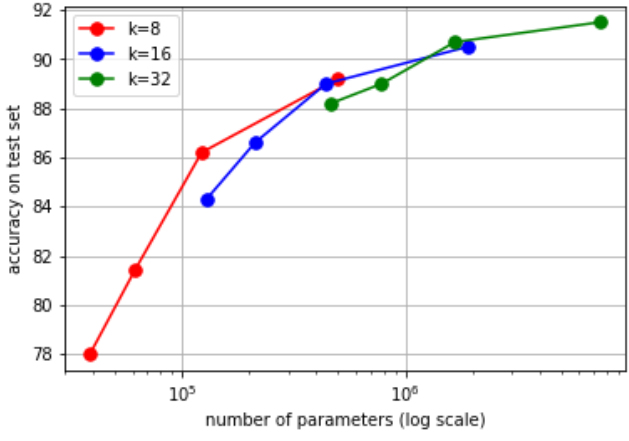

Fig. 15. Accuracy depending on the number of parameters (results grouped by depths)

4.3 Comparison with other networks

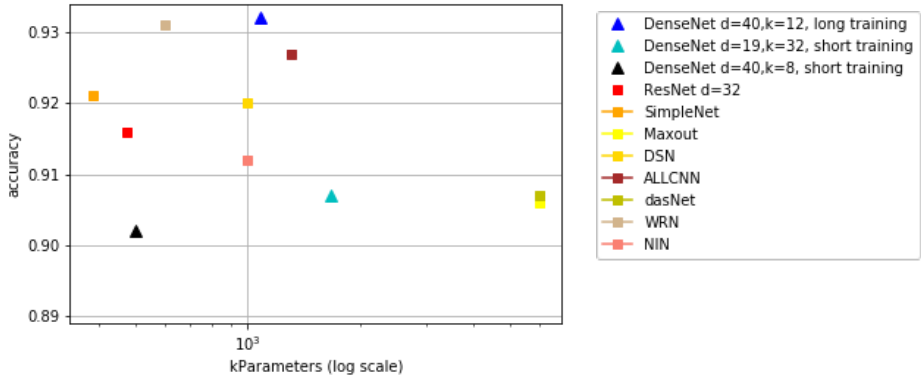

Fig. 16. Comparison of DenseNet with other CNN networks

Figure 16 has to be considered with caution. Most of the DenseNet networks of this project have been trained with only 45 epochs so they probably did not reach their optimal accuracy. However one can see that they are not so far from the other networks, and they use less parameters than the less compact networks. When a DenseNet network is trained with a lot of epochs (DenseNet d=40, k=12, long training) it performs very well reaching a 0.932-accuracy, which is more than the other networks presented in the paper.

Entire names of the networks can be found in the original paper.

5 Conclusion

DenseNet, thanks to its structure, theoretically solves issues other CNN networks face. The tests we conducted materialized the expectations from theory, with an 0.932-accuracy for a 40 layers network. Different settings (depth and growing rate) may be used depending on what one wants to optimize. A small depth and large growing rate seems to reduce the training time whereas a large depth and a small growing rate seems to reduce the number of parameters. However this observation has been made on few simulations so it would be inappropriate to generalize this conclusion to a large extent.

References

1. Huang, Liu, van der Maaten, Weinberger: Densely connected convolutional networks
2. Krse, van der Smagt: An introduction to neural networks
3. Goodfellow, Smith, Courville: Deep learning book
4. Hasanpour, Rouhani, Fayyaz, Sabokrou: Let's keep it simple, using simple architectures to outperform deeper and more complex architectures