Deep Convolutional Neural Networks
Alleviating the Degradation Problem

Christos Dimopoulos - s1687053

May 21, 2017

1 Introduction

During the last years, image classification tasks have been treated effectively with deep convolutional neural networks [1, 2, 6, 14]. This kind of networks take into account a number of levels of features and classify images accordingly, where the number of layers they consist of enrich the features that could be extracted. Many significant research contributions on image recognition datasets i.e. CIFAR 10/100, ImageNet [3, 11, 15], take advantage of very deep models usually over 16 to hundreds of layers. However, it is naturally difficult to train deep neural networks with the common gradient based techniques since the gradients of the weights become extremely small after a certain number of layers. The gradient vanishing problem have been greatly addressed and the networks become able to converge with SGD by proper initialization [5, 7, 8] and normalization of the input of each layer [13].

While the networks become able to converge, it is observed that the increase in number of layers leads to saturation of their accuracy. Interestingly, this phenomenon is not caused due to overfitting and stacking more layers leads to higher training error as reported in [16, 17] and verified by the experiments of this report. The phenomenon of higher training error in deep neural networks is called *the degradation problem*. In this paper, simple deep convolutional architectures are presented which reveal the degradation problem and more advance *state of the art* architectures which solve the degradation problem are being studied. At first, experiments are conducted with the VGG network and plain deep CNN in order to show the degradation of accuracy while the depth of each network increases. In addition, advanced architectures, ResNet, HighwayNet and DenseNet [4, 12, 17], which counter the aforementioned problem are being studied.

The data sets which were used for the experiments are CIFAR-10 and CIFAR-100. CIFAR-10 consists of 60 thousand color images of size 32x32 pixels and three RGB color channels (total dimension is 3072) as it is shown in figure 1. The images are represented by 10 classes of different objects i.e. cat, dog, airplane etc. The dataset is split in 40 thousand images for the training set and 10 plus 10 thousand images for the validation and test sets. CIFAR-100 is similar in size to CIFAR-10 but instead of 10 classes it consists of 100 classes of different images with 20 super-classes of the aforementioned 100. In addition, CIFAR- 100 is split similarly to CIFAR-10 for training, validation and test sets.

2 Degradation in Deep Convolutional Networks

One could assume that by giving a neural network more parameters, it should be able to fit the train data at least as well as before. This hypothesis that an increase in numbers of layers could enrich the feature extraction and representation in neural networks, comes with a surprising counter effect. This effect is not caused by overfitting or by vanishing of the gradients [5, 8] as one could expect since many methods such as batch normalization cope effectively with this problem [13]. The effect is caused by the complexity of the model where the gradient based methods find difficulties in converging on very deep and complex architectures [5, 7, 9]. This effect is called degradation problem in deep neural networks. It is hard or yet impossible to prove it mathematically [4, 16, 17] and thus, it is only rather intuitive and could be observed experimentally.

To prove the degradation of accuracy in deep convolutional neural networks, we experiment with two popular CNN architectures, VGG and plain Deep CNN, in order to show that it is not just a problem of only one of them but rather a uniform one. We begin by referencing the baseline

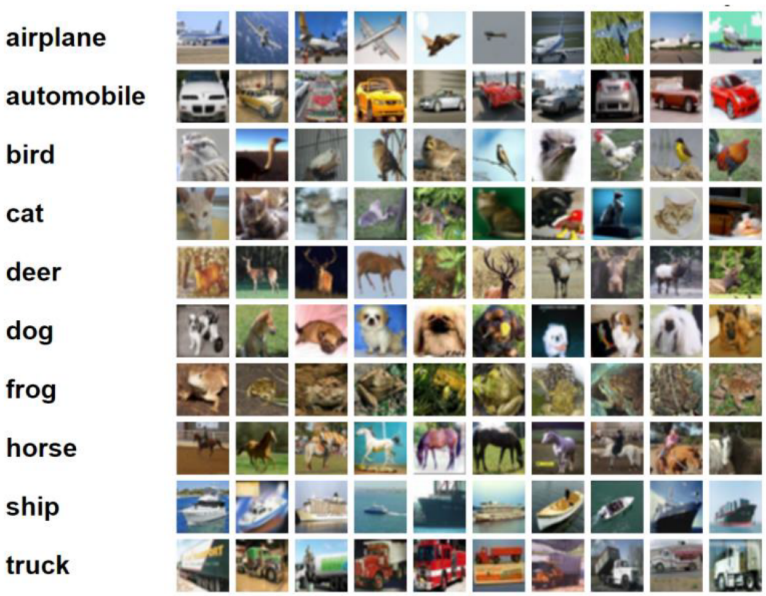

Figure 1: Example images from CIFAR-10 dataset.

model which was created at the previous coursework. Although, that model was only a multilayer feedforward neural network and did not have such a high complexity as the ones we will present in this section and later in the paper, the degradation problem could also be observed while increasing the depth of it.

2.1 Baseline Model

In the previous coursework, we built a baseline feed forward neural network where we focused more on the advanced regularization methods of neural networks [2] and try to solve the problem of gradient vanishing with batch normalization [13]. Although, these research questions and the feed forward neural network are not highly relevant with the degradation problem in deep convolutional networks which we are trying to encounter, it is meaningful to use it as a baseline model for accuracy comparison to more advanced convolutional architectures and sufficient to prove the degradation problem while increasing the depth even in simple models such as this one.

2.1.1 Architecture and Outline of Experiments

The baseline model, as introduced in the previous coursework, is a feedforward neural network. Each feed forward layer has width equal to 600 which was found optimal through experiments and the activation function is ReLu by using tf.nn.relu in tensorflow. The weights are initialized as in [8] which is proposed to capture the non-linearities of the rectified linear unit, take into account the scaling of the variance in each input layer and help convergence of deep neural networks by countering the gradient vanishing problem. We implemented this initialization by using tf.contrib.layers.variance_scaling_initializer() in Tensorflow. The biases were initialized to zero as it is most common. Before the non-linear activation of each layer, the outputs were normalized with batch normalization [13] by using tf.contrib.layers.batch_norm(), where we used a weight decay equal to 0.9 so that the global variance would begin updating at the start of each experiment.

To counter overfitting, we use dropout by tf.nn.dropout() after each layer with dropping probability equal to 0.2 which optimized through experiments. Dropout randomly drops units after each activation in order to prevent high co-adaptation of the network. Another method of overfitting that we applied to the network is L2 regularization. We take the second norm of the weights of each layer in the network and add them to the final error function penalized by the regularization parameter which was set equal to 0.0001 through experiments. L2 regularization was implemented with tf.nn.l2_loss() in tensorflow. The output of each model is passed to a simple feedforward layer which outputs the desired predictions.

To show the degradation in the baseline model of the previous coursework, we experiment with the aforementioned model for different number of hidden layers, in particular, 1,3,5,8 and 16. We train the model with momentum optimizer (tf.nn.train.MomentumOptimizer) with learning rate 0.01 and momentum 0.9 as it is proposed in the majority of the research papers [1, 4, 14, 17]. We track the error and accuracy for the training and validation sets which were implemented by tf.nn.softmax_cross_entropy_with_logits() and tf.cast(tf.equal(targets,outputs)) respectively.

2.1.2 Results and Discussion

Figure 2 shows the training and validation error of the baseline model for CIFAR-10 and CIFAR-100. For both datasets the error on training and validation sets is less when the depth increases over 3 hidden layers. From the validation figures we could be convinced that this does not happen due to overfit since the error declines until it saturates and remains constant. We would expect that the models with more hidden layers would achieve lower error, but the experiments show that the optimal number of hidden layers is three and any increase in the depth actually achieves higher error than the model with one hidden layer as the example of 14 hidden layers show in figure 2.

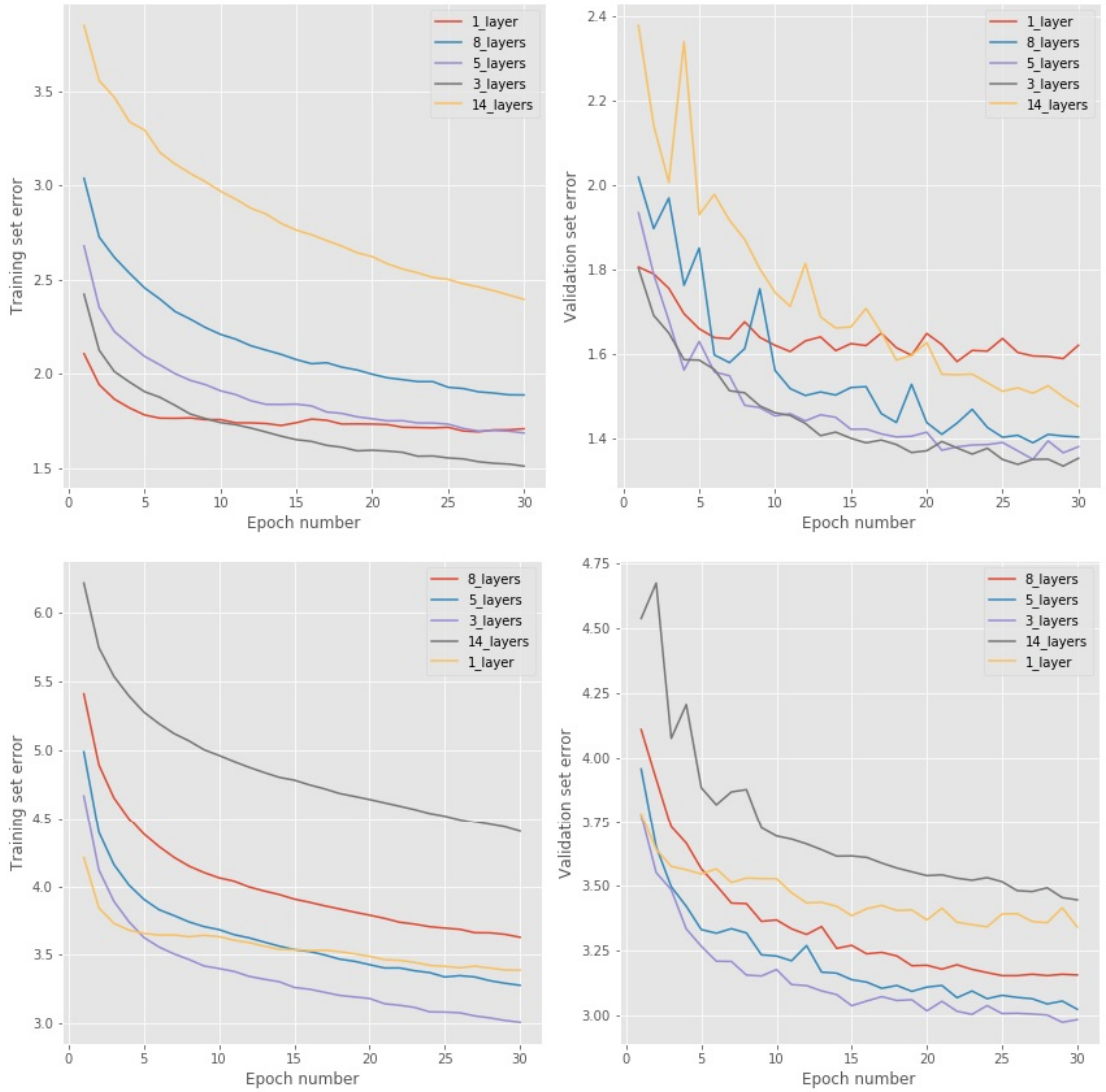

Figure 2: Training and validation error figures for the baseline model. **Top**: CIFAR-10, **Bottom**: CIFAR-100

The same conclusion we could draw from the accuracy metrics of table 1. The higher accuracy achieves the model with 3 hidden layers and every other model with more than 3 hidden layers achieves lower accuracy and thus, the accuracy degradation becomes clear. When we increase

the number of hidden layers the accuracy declines instead of increasing. At the same time, this degradation does not happen due to overfitting as figure 2 suggests. The validation error follow the downward trend of the training until it saturates to a constant value and does not extrapolate or start increasing as it could happen due to overfitting.

Nr. of Hidden Layers	Accuracy CIFAR-10 (%)	Accuracy CIFAR-100 (%)
1	44.6	21.3
3	52.9	28.1
5	52.4	26.8
8	51.4	23.8
14	47.8	17.9

Table 1: Validation Accuracy of the baseline model for CIFAR-10/100.

2.2 VGG Network

The VGG network is one of the most popular architectures of deep convolutional neural network architectures for image recognition. Its popularity was based on its unique architecture which made it possible to exploit the depth of convolutional neural networks and achieved higher accuracy than the other CNN architectures since it was proposed [3]. In particular, the VGG network combined efficiently convolutional layers and max pooling layers which made it possible to increase the depth of the network. However, this increase was limited to 19 layers and the reason for that is the complexity of the network that caused degradation. In the rest of the section we will reproduce the experiments for the VGG network for CIFAR-10/100 datasets and show the degradation effect.

2.2.1 Architecture and Outline of Experiments

The core of the VGG network consists of stacks of convolutional layers with the same feature map size and number of convolutional layers and after each stack there is a max pooling layer which is responsible for the discretization of the input. In particular, the objective of the max pooling is to down-sample an input representation by reducing it's dimensionality and allowing for assumptions to be made about features contained in the sub-regions. The max pooling layer is an important part of the network and the main reason that enabled it to grow deep up to a certain point. It reduces the computational cost by reducing the number of parameters to learn and thus makes it simpler to converge while increasing its depth.

The depth of the network is realized by the increase of the number of convolutional layers in each stack. Overall, we have 3 stacks of convolutional layers of 16, 32 and 64 feature map sizes (number of channels) respectively where each stack has as input an image of size 32, 16, and 8 which happens due to the downsampling of the max pooling layer between each stack. The convolution layer was implemented by using the tf.slim.conv2d() function of the slim library in tensorflow which provides a more abstract implementation of the layer and makes it more concise to use, whereas, the max pooling layer was implemented by tf.nn.maxpool with a 2x2 stride in order to downsample the image in half. An overview of the architecture of each model is presented in table 2.

Each convolutional layer has a very small receptive field of 3x3 as it is suggested in the majority of the architectures [1, 3, 4, 17]. The 3x3 kernel or receptive field size was chosen small and not bigger because small convolutional layers incorporate more rectification (through the non-linearity) than one convolution with larger kernel size. At the same time, they introduce less parameters which make the overall network simpler and easier to optimize [3]. After each convolutional layer, we apply batch normalization, as in section 2.1 of the baseline model, in order to achieve faster convergence and alleviate the gradient vanishing [13]. The normalization of each convolutional layer follows the rectified non-linear function as the authors suggest in [3]. The weights of each layer were initialized with the scaling variance initialization to better capture the non-linearities and help convergence without pre-training [8] by using tf.contrib.layers.variance_scaling_initializer().

The convolutional stacks are followed by two fully connected layers of width 1024 where each of them is regularized by dropout with dropping probability equal to 0.5 as suggested in [3]. The predictions were drawn from a fully connected layer for which we apply soft max operation as we

8 Layers	14 layers	20 Layers
input 32 x 32 RGB image		
		conv3-16
		conv3-16
	conv3-16	conv3-16
	conv3-16	conv3-16
conv3-16	conv3-16	conv3-16
conv3-16	conv3-16	conv3-16
max pooling		
		conv3-32
		conv3-32
	conv3-32	conv3-32
	conv3-32	conv3-32
conv3-32	conv3-32	conv3-32
conv3-32	conv3-32	conv3-32
max pooling		
		conv3-64
		conv3-64
	conv3-64	conv3-64
	conv3-64	conv3-64
conv3-16	conv3-64	conv3-64
conv3-64	conv3-64	conv3-64
max pooling		
FC-1024		
FC-1024		
Soft-max Output		

Table 2: VGG configuration. The convolutional layer parameter are denoted as "conv<receptive field size>-<number of channels>".

did in section 2.1. For every convolutional and fully connected layer we apply a weight decay of 0.0001 by adding their L2 regularizers (tf.nn.l2_loss) to the error function.

We experiment over the depth of the VGG network as shown in the configuration of table 2 and suggested in [3]. The models are trained with mini-bacth size of 50 and momentum optimizer with learning rate 0.1 and momentum 0.9 where the learning rate is decayed every 10 epochs in order to help the optimizer converge. We track the error and accuracy for the training and validation sets which were implemented as in section 2.1 respectively. The experiments were conducted for 30 epochs which proved to be enough to show the degradation of the network.

2.2.2 Results and Discussion

The results of the experiments with the VGG network make clear the degrading performance while the depth of the network increases. As it was assumed, the higher error of the deeper models is not due to overfitting as we could derive from figure 3 since the error of each model still declines. Furthermore, in both training and validation sets the error of the 20 layers model is approximately 0.25 to 0.5 higher than the one with 16 layers for both datasets. The same seems to be the case between 16 and 8 layers models. Table 3 presents the accuracy for each VGG model for both CIFAR-10 and 100 datasets. The highest accuracy was achieved for the 8 layers model at 63% and 32% respectively. Ultimately, the lower accuracy of the models with more layers confirm the degradation hypothesis.

On another note, the VGG network achieves a substantially better performance than the baseline model of section 2.1. The 8 layers VGG network achieves 32% accuracy which is 4% more than the baseline feed forward network with 3 layers. The reason for this is because of two main factors. The first is the advanced architecture of the VGG network which capitalizes the feature map representations of the convolutional layers and makes it possible to extract more meaningful latent features of the images. The second is the increased depth of the VGG network which is composed by 8 layers in contrast to 3 of the baseline model which allows to take into account more levels of features.

Nr. of Hidden Layers	Accuracy CIFAR-10 (%)	Accuracy CIFAR-100 (%)
8	63	32
16	61.25	28.8
20	48.9	22.7

Table 3: Validation Accuracy of the VGG network for CIFAR-10/100.

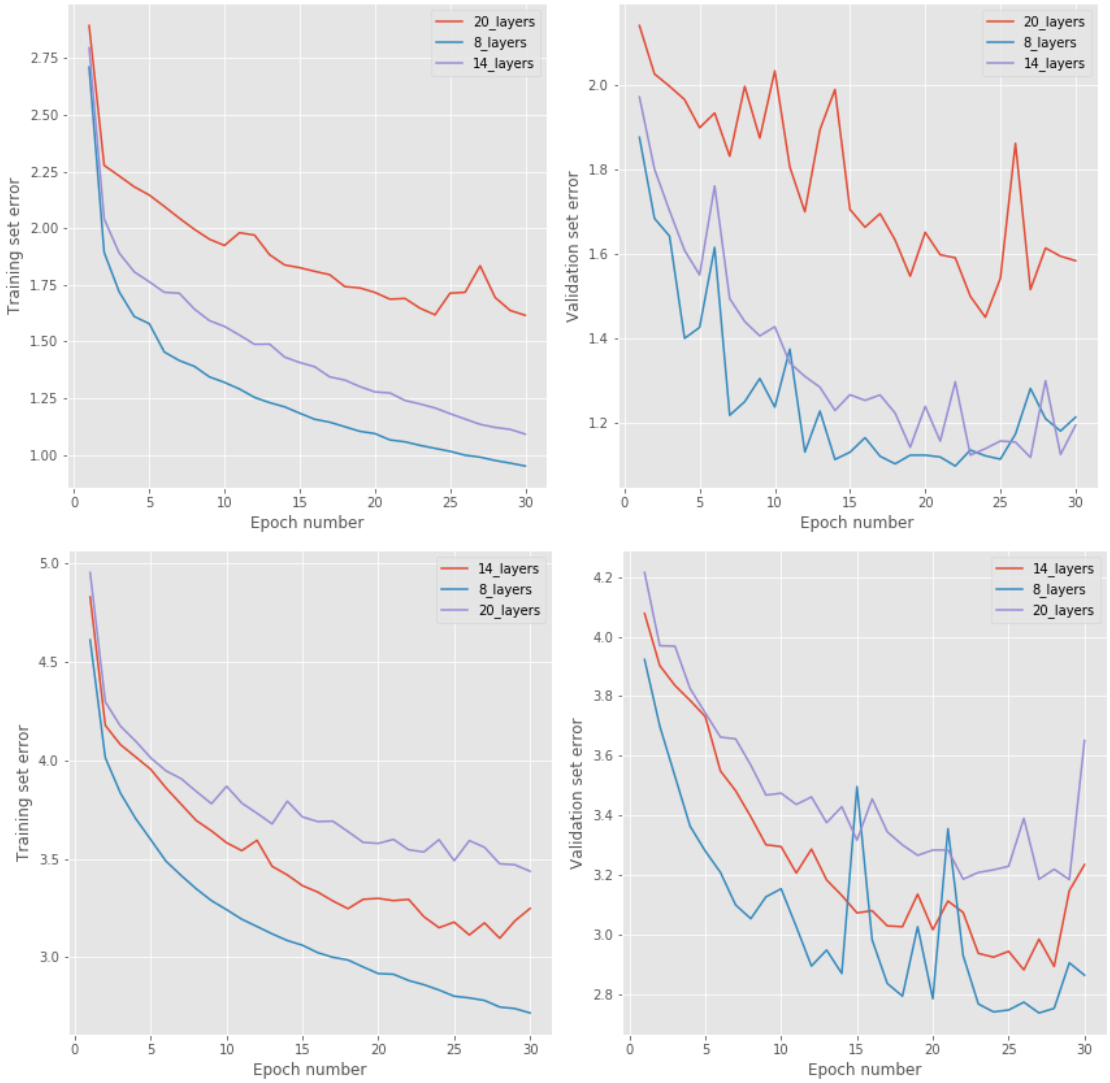

Figure 3: Training and validation error figures for the VGG network. **Top**: CIFAR-10, **Bottom**: CIFAR-100

2.3 Plain Deep Convolutional Neural Network

To further show how the reduced performance in deep convolutional neural networks, we will implement and experiment with a plain deep convolutional neural network. This network mimics the overall structure of the more advanced architectures that will be presented in section 3 but misses the techniques which allow the network to surpass the degradation problem and show improvement in accuracy with the increase in the depth of the network. This plain deep convolutional neural network was presented in [4] and was used as a baseline model for showing the degradation problem and for comparison with architectures that alleviate it.

2.3.1 Architecture and Outline of Experiments

The architecture of the plain deep convolutional network follows the form in figure 6. The first layer is a 3x3 convolutional layer with input channel size (filters) of 16 which takes as an input an image of size 32x32x3. Then we use a stack of *6n* 3x3 convolutional layers with feature maps of size [32,16,8] respectively with *2n* layers for each feature map size. In this architecture, the subsampling of the images is realized by 3x3 convolutions with stride 2x2 which essentially works as a max pooling layer but keeps the features representations instead of taking the maximum samples [4]. The output of the convolutional stacks is forward by a global average pooling layer to a 10 or 100 -way fully connected softmax output layer for CIFAR-10 and CIFAR-100 respectively. In total, the network has *6n+2* stacked weighted layers and table 4 summarizes the architecture.

output map size	32x32	16x16	8x8
# layers	1 + 2n	2n	2n
# filters	16	32	64

Table 4: Plain deep convolutional network architecture.

Each convolutional layer has a very small receptive field of 3x3 which introduce less parameters and makes the overall network simpler and easier to optimize as it is suggested in the majority of the architectures [1, 3, 4, 17] and is implemented with slim.conv2d() in tensorflow. After each convolutional layer, we apply batch normalization in order to achieve faster convergence and counter the gradient vanishing effect [13]. The normalization of each convolutional layer follows a ReLu activation function as the authors suggest in [4] and is gonna be followed for every architecture in this paper. The weights of each layer were initialized with the scaling variance initialization which further helps convergence [8]. For every convolutional layer we apply a weight decay of 0.0001 by l2 regilarization (tf.nn.l2_loss) which is added to the error function.

We compare *n=[3,5,7]*, leading to 24, 34 and 48-layers networks. The models are trained with minibatch size of 50 and momentum optimizer with learning rate 0.1 and momentum 0.9 where the learning rate is divided by 10 every 10 epochs where the accuracy saturates to a constant in order to help convergence as suggested in [4]. We track the error and accuracy for the training and validation sets which were implemented as in section 2.1 respectively. The experiments were conducted for 30 epochs.

2.3.2 Results and Discussion

The validation curves in figure 4 show that the 34-layers and 48-layers plain nets have higher validation error. To reveal the reasons for this, we compare it with the error curves of the training procedure. The degradation problem could be observed - the deeper 34 and 48-layers networks show higher training error, even though the solution space of the 20-layer network is a subspace of them. We are able to argue that this optimization difficulty is not caused by vanishing gradients since the networks are trained with batch normalization, which ensures forward propagated signals to have non-zero variances [13]. We could also verify that the backward propagated gradients exhibit healthy norms, so neither forward nor backward signals vanish [4]. In fact, the 34-layer plain network still achieves competitive accuracy, indicating that the optimizer works to some extend.

Nr. of Hidden Layers	Accuracy CIFAR-10 (%)	Accuracy CIFAR-100 (%)
24	64.7	31.6
34	55.4	23.9
48	45.5	20.4

Table 5: Validation Accuracy of the plain convolutional network for CIFAR-10/100.

The plain deep CNN architecture in comparison to the VGG network with 8 layers achieved 1.7% higher accuracy for the model with 24 hidden layers. This supports the hypothesis that the increased depth in neural networks should achieve higher accuracy. However, the networks error still degrades while the number of layers increase as figure 4 presents. Without the networks to

overfit, the error of the 48-layers network is less by 2.5 than the 20-layers one at the stable state whereas the 34-layers plain net is less by 0.5.

Table 5 clearly exhibits the accuracy degradation effect which happens while the network grows deeper. The model with 24 layers achieves accuracy of 64.7%, whereas, the 34-layers and 48-layers-models achieve 10% and 20% less accuracy respectively. The model with 48 layers has substantially higher error than the other models which could be observed in figure 4 for both datasets. This indicates the optimization difficulty of the deeper network to converge due to its complexity.

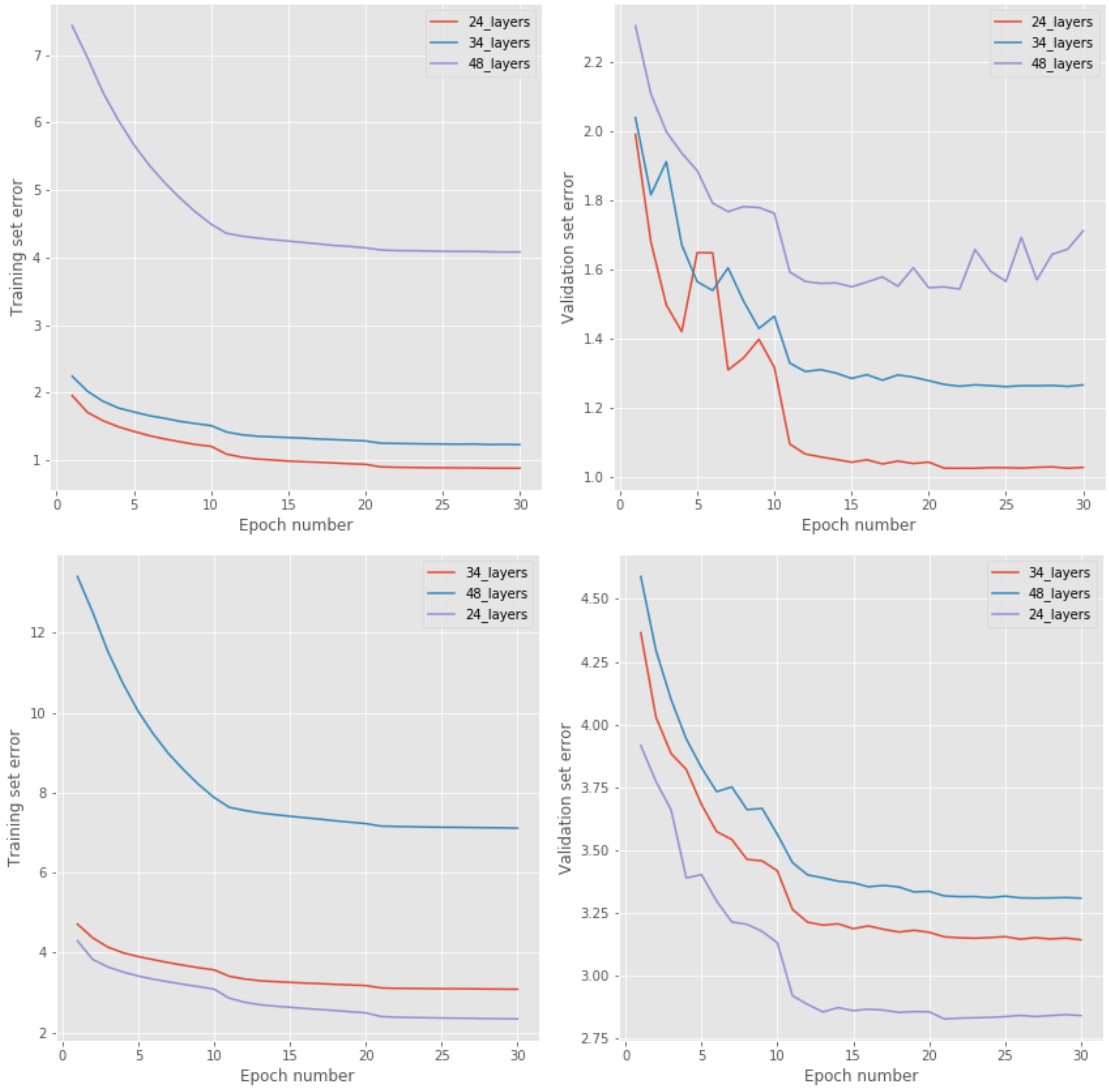

Figure 4: Training and validation error figures for the plain convolutional network. **Top**: CIFAR-10, **Bottom**: CIFAR-100

3 Alleviating the Degradation Problem

There is plenty of theoretical and empirical evidence that the depth of neural networks is a crucial component for their success [17]. However, as suggested and proved in section 2, training becomes rather difficult with increasing depth because of the degradation effect in very deep models. The current solvers fail to optimally converge while the networks become substantially deep due to the high complexity of their weight space. Recently, there has been a number of breakthrough architectures in deep convolutional neural networks that succeeded in image recognition applications with high accuracy while preserving high depth and efficiently cope with the degradation problem [4, 11, 12, 17, 18]. In this paper, we will present and experiment with Residual Networks (ResNet) [4], Highway Networks [17] and Densely Connected Convolutional Networks (DenseNet) [12].

These *state of the art* architectures introduce special design characteristics such as skip or short connections and residual representations which allows the optimizer to find identity mappings in deeper architectures that help it to converge as in more shallower networks. By this way, the networks gain higher accuracy from greatly increased depth and substantially better results than the networks of section 2. In the next sections, we will present the aforementioned networks in terms of architecture and the different design options they introduce to alleviate degradation. At the same time, we will try to reproduce the experiments designed in the papers and show the increased performance they achieve for CIFAR-10 and CIFA-100 datasets.

3.1 Highway Networks

The Highway network architecture was one of the first to introduce effective network optimization with *virtually* high arbitrary depth. The network was inspired by Long Short Memory Recurrent neural networks [19] and it makes use of a gating mechanism which regulates how the information flows throughout the network [17]. This gating mechanism creates paths, which are called information high-ways, that allow the signal to flow across many layers without being depreciated.

In a typical feedforward neural network, the output y of each hidden layer would apply an affine transformation followed by a non-linear transformation H on its input x, as $y = H(x, W_H)$, where W_H are the weight parameters of the hidden layer. The highway network introduces two more non-linear transformations $T(x, W_T)$ and $C(x, W_C)$, where T is referred as the transformed gate and C as the carrier gate and they are combined as $C = 1 - T$ for simplicity. Then the output could be re-written as $y = H(x, W_H) * T(x, W_T) + x * (1 - T(x, W_T))$ which leads to,

$$y = \begin{cases} x & \text{if } T(x, W_T) = 0 \\ H(x, W_H) & \text{if } T(x, W_T) = 1 \end{cases} \tag{1}$$

The dimensionality of the input x, output y and transformations $H(x, W_H)$ and $T(x, W_T)$ should be the same in order for the equation 1 to be valid. Thus, a highway layer can decide the signal path between that of a non-linear layer and that of one which simply passes its input to the output [17].

3.1.1 Architecture and Outline of Experiments

The implementation of the highway network follows the same pattern as the plain convolutional network of section 2.3. We split the architecture into highway stacks of convolutional layers where each stack consists of a highway convolutional layer H and a transfer convolutional layer T. Following equation 1, the output of each highway stack is defined as $y = H * T + x * (1 - T)$. It is mentioned that the input, output and non-linear layers should have the same dimensionality for the above equation to be valid. In the case where it is desired to change the dimensionality of the stack, we augment the input with zero padding in order to match the dimensions of the new convolutional stack. This was implemented in tensorflow with tf.pad([0,0],[0,0],[0,0],[new_size,new_size]).

The 32x32x3 input image is forwarded into the first 3x3 convolutional layer with input filter size of 16. Then we use a stack of *6n* 3x3 convolutional layers with filters of sizes [32,16,8] respectively with *2n* layers for each filter size. In this architecture the subsampling of the images is realized by 3x3 convolutions with stride 2x2 which samples the image into lower dimensions but keeps the features representations. The last 3x3 convolutional layer has output of 10 which is forwarded through a global average pooling layer to a 10 or 100 -way fully connected softmax output layer for CIFAR-10 and CIFAR-100 respectively. Table 6 summarizes the highway architecture.

24 Layers	34 layers	48 Layers
input 32 x 32 RGB image		
conv3-16		
conv3-16 conv3-16 } $x3$	conv3-16 conv3-16 } $x5$	conv3-16 conv3-16 } $x7$
conv3-16, stride 2x2		
conv3-32 conv3-32 } $x3$	conv3-32 conv3-32 } $x5$	conv3-32 conv3-32 } $x7$
conv3-32, stride 2x2		
conv3-64 conv3-64 } $x3$	conv3-64 conv3-64 } $x5$	conv3-64 conv3-64 } $x7$
conv3-10/100		
global average pooling		
Soft-max Output		

Table 6: Highway network configuration. The convolutional layer parameter are denoted as "conv<receptive field size>-<number of channels>".

Each convolutional layer is implemented by using slim.conv2d() in tensorflow. Following the implementation in [17], the output of each convolutional layer is normalized with batch normalization [13] to ensure that the forward propagated signals have non-zero variances and gradually vanish. Each layer is activated with the rectified linear function (ReLu) as in every experiment so far. The weights of every layer were initialized with scaling variance as in [8] to achieve more accurate and faster convergence. The authors suggest that the biases of the layers should be initialized with negative value as it was proved sufficient for various zero-mean initial distributions of the weights and many activation functions [17]. In the upcoming experiments for the highway networks, the biases were initialized to -1.0. The network is regularized by L2 regularization (tf.nn.l2_loss()) to avoid overfitting with weight decay equal to 0.0001.

We experiment over the depth of the highway network with *n=[3,5,7]*, leading to 24, 34 and 48-layers networks which are the same depths with the plain network for comparison purposes. The models are trained with minibatch size of 50 and momentum optimizer with learning rate 0.1 and momentum 0.9 where the learning rate is divided by 10 every 10 epochs to help the optimizer get closer to the minimum. We track the error and accuracy for the training and validation sets which were implemented as in section 2.1 respectively. The experiments were conducted for 30 epochs.

3.1.2 Results and Discussion

The results for CIFAR-10 dataset indicate that learning to route information through neural networks has improved the scaling of depth. While the 24-layer network achieves the lowest error, the two deeper models achieve similar results as figure 5 presents. An important note is that the models overfit since the training error figure declines while the validation error gets higher. This indicates that a higher regularization penalty should have been used to avoid it. The accuracy results of table 7 confirm that the highway network is able to cope with the accuracy degradation. All of the three models achieve accuracy above 65%. In comparison with the plain net of section 2.3 and table 5, the 34-layer model achieves 10% higher accuracy while the 48-layer model achieves better performance by 20%.

Nr. of Hidden Layers	Accuracy CIFAR-10 (%)	Accuracy CIFAR-100 (%)
24	67.7	39.2
34	65.4	39.9
48	65	30.07

Table 7: Validation Accuracy of the Highway network for CIFAR-10/100.

On the other hand, the error curves for CIFAR-100 still show signs of degradation. In particular, the 48-layer network exhibits 0.75 higher error than the 24 and 34-layers networks throughout the training and validation time. However, the 34-layer model exhibits almost the same performance as the 24-layer network. In terms of accuracy, we get the best performance for the 34-layer model at

39.9 % which is 17% more than the equivalent model in the plain net architecture while the 24-layer highway model achieves 39.2% accuracy which is 7.6% more than the 24-layer plain network. To conclude, the highway architecture with the alternative route of information that proposed made it possible to cope to some extend with the degradation effect and grow deeper networks which lead to an overall increase performance of 7%.

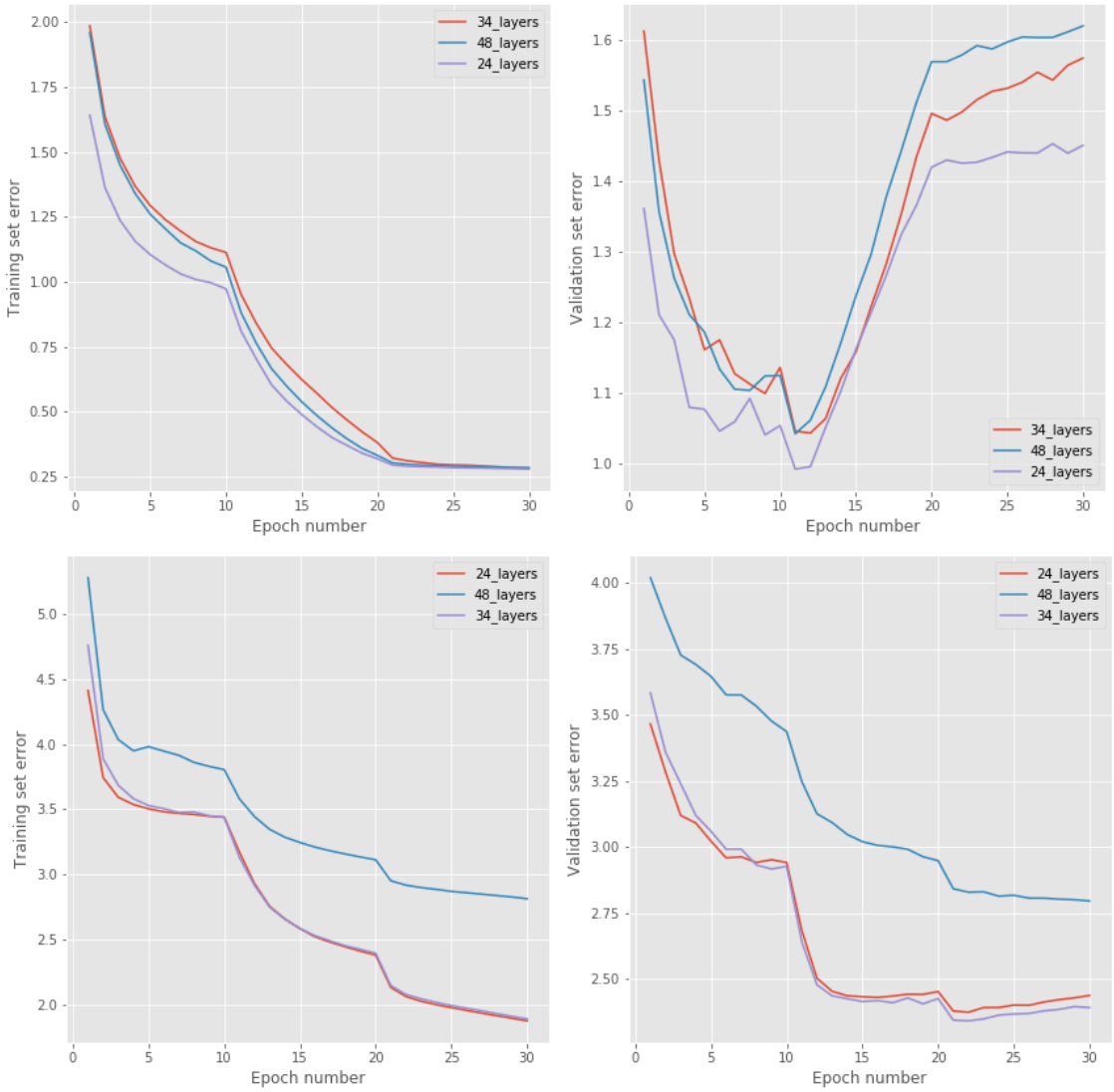

Figure 5: Training and validation error figures for the Highway network. **Top**: CIFAR-10, **Bottom**: CIFAR-100

3.2 Deep Residual Convolutional Networks

Deep residual convolutional neural networks or *ResNets* is one of the most recent breakthrough convolutional neural network architectures that achieves high performance on image recognition tasks by capitalizing the advantages of very deep neural networks [4]. ResNet addresses the degradation phenomenon by introducing residual mappings in each layer of the network. Conceptually, it consists of *shortcut connections* between stacks of non-linear layers which could be mapped as $F(x) = H(x) - x$, where $H(x)$ is the underlying original mapping and $F(x) + x$ is the residual one. The authors in [4] argue that it is easier to optimize the residual mapping since it is less complex to push the residuals to zero than to fit an identity mapping. In figure 6, shortcut connections are presented with arrows that skip blocks of convolutional layers. The output of the shortcut connection is added to the output of a convolutional stack by simply performing an identity mapping. An important advantage of the shortcut connections is the fact that they do not pose more

computational complexity to the network since they do not add any extra parameters.

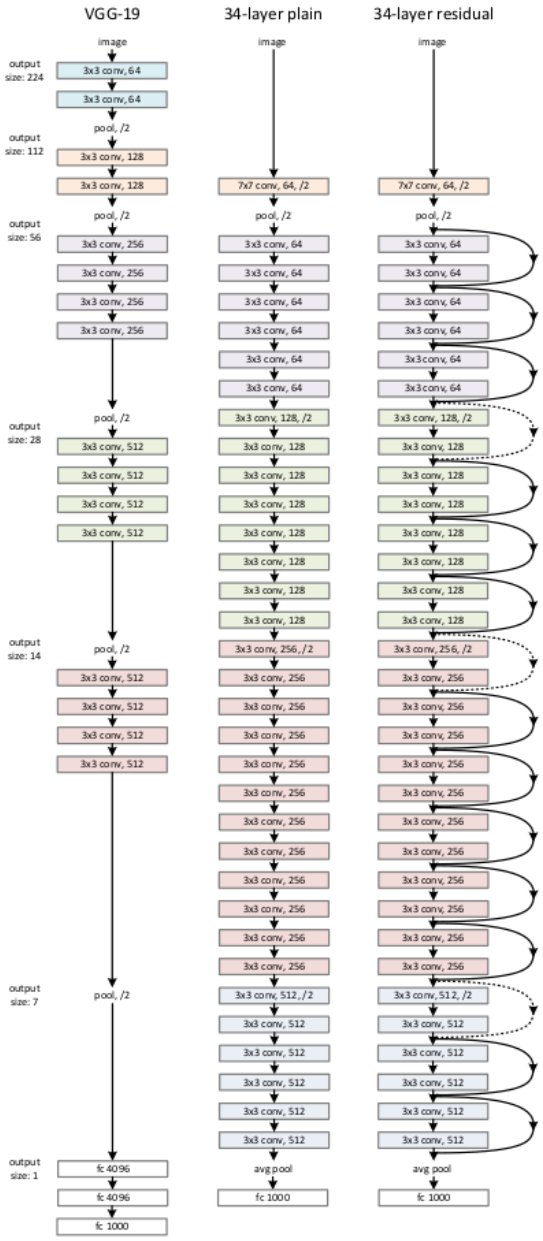

Figure 6: Example network architectures for Imagenet. **Left**: VGG 19 layers network. **Middle**: plain 34 layers network. **Right**: Residual 34 layers network. **Source**: [4]

The parameter-free property of the shortcut connections that ResNets introduce gives them a clear advantage over the highway networks of the previous section, since their information gates depend on the input and have learning parameters [17]. In addition, the residual mappings of ResNets are never closed and the information always flows through the network, in contrast to the highway networks where the signal gates could be closed if they are approaching zero (eq.1). At the same time, as the highway networks still suffer to some extend from degradation as it was proved for CIFAR-100 dataset (figure 5).

3.2.1 Architecture and Outline of Experiments

The architecture and implementation of the residual network follows the example form of figure 6. With similar strategy as the previous deep convolutional architectures, the network is implemented by stacks of convolutional layers with the same feature map size across each stack. The architecture

according to [4] follows two rules in order to preserve the same complexity in each stack, a) the layers in each stack have the same number of filters and b) when the feature map is halved, the number of filters is doubled. As with the highway network architecture, the downsampling is performed by convolutional layers of stride 2 which samples the image into lower dimensions but keeps the feature representations. Stacks of *6n* 3x3 convolutional layers with filters of sizes [32,16,8] are used respectively with *2n* layers for each filter size.

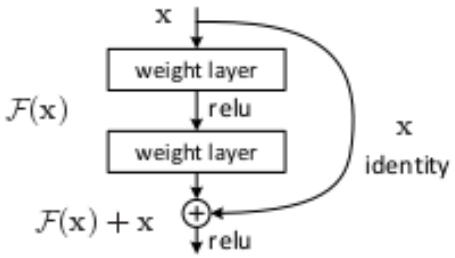

Figure 7: Residual Block. **Source**: [4]

Figure 7 outlines the basic structure of a residual block. The input of each block is forwarded and added to the output convolutional layer. The residual mapping as well as the output of each convolutional layer before it passes through a ReLu activation function (tf.nn.relu). In this architecture the subsampling of the images is realized by 3x3 convolutions with stride 2x2 which samples the image into lower dimensions but keeps the feature representations. A global average pooling layer (layers.conv.global_avg_pool) is used, which pools the output of the convolutional stacks to the desired dimensions and passes the signal to the output softmax operation. Each convolutional layer is implemented in tensorflow with slim.conv2d() and the weights were intialized with the scaling variance initialization,as in previous sections, which better accumulates the non-linearities in the data and helps convergence [8]. The weights of the network are regularized with L2 norm and L2 penalty equal to 0.0001 as in the previous architectures throughout the paper. The output of each convolutional layer is normalized with batch normalization (contrib.layers.batch_norm) which centers the data and help the gradients to avoid taking extreme values [13].

The identity residual mappings, solid lines in figure 6, are valid when the input and output of the block have the same size. However, the connections when the dimensions increase, dotted line shortcuts in figure 6, the addition is not valid. To implement this discrepancy, the authors in [4] suggest two options: a) the identity mapping is reinforced with zero paddings to increase dimensions as in section 3.1 of the highway networks. b) The projection shortcut could be realized with 1x1 convolutions. The first option is parameter free and adds no extra complexity to the network but in the second option the network could learn and project more efficiently the dimensionality increase in each stack.

We experiment over the depth of the ResNet with *n=[3,5,7]*, leading to 24, 34 and 48-layers networks so we could compare with the exact deep models of the previous architectures. In addition, the experiments were conducted over the two shortcut connections options, which were mentioned above, since such an implementation decision could play a role on the performance of the model while growing deeper. The models are trained with minibatch size of 50 and momentum optimizer with learning rate 0.1 and momentum 0.9 where the learning rate is divided by 10 every 10 epochs to help the optimizer get closer to the minimum. We track the error and accuracy for the training and validation sets which were implemented as in section 2.1 respectively. The experiments were conducted for 30 epochs.

3.2.2 Results and Discussion

The results of the experiments over the depth of the ResNet confirms that the proposed architecture is able to cope effectively with the degradation problem. The deep 48-layer model achieves the lowest error compared to the less deep 24 and 34 layers residual networks as it could be observed in figure 8. The information in the present residual architecture could efficiently pass through the identity mappings which helps the stochastic gradient descent solver optimize effectively the error function. This enabled the network to grow deeper and realize better feature extraction. The accuracy metric of table 8 could confirm the success over degradation. For both CIFAR-10 and

CIFAR-100 datasets, the model with 48-layers achieves the highest accuracy at 70% and 40.1% respectively.

Nr. of Hidden Layers	Accuracy CIFAR-10 (%)	Accuracy CIFAR-100 (%)
A-24	68.4	39.9
A-34	69.3	40
A-48	68.9	40.1
B-24	68.7	39.9
B-34	70.1	39.9
B-48	70.2	40.1

Table 8: Validation Accuracy of the ResNet for CIFAR-10/100 over depth for two shortcut options A & B.

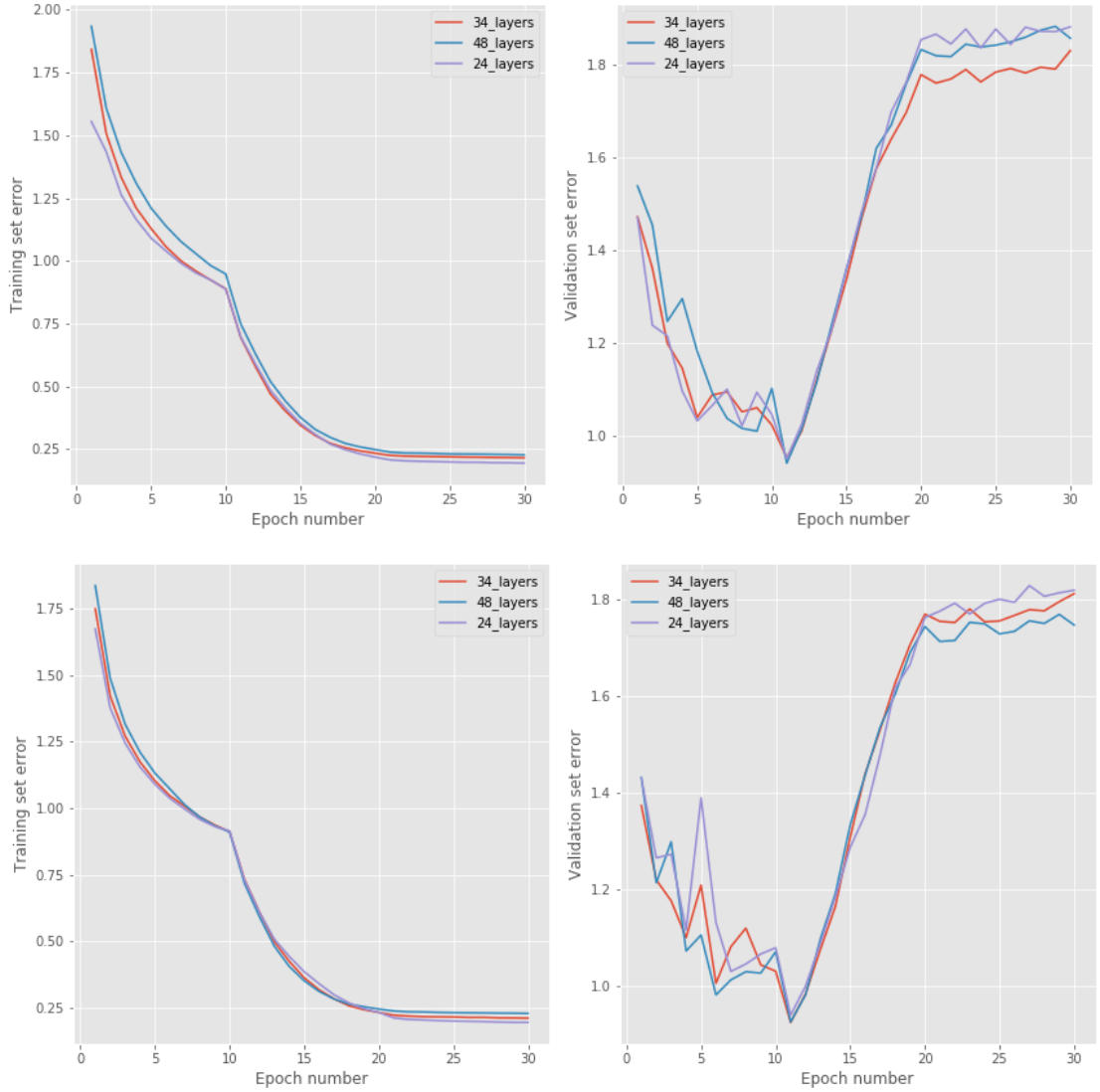

Figure 8: Training and validation error figures for the Residual network on CIFAR-10. **Top**: Shortcuts Option-A, **Bottom**: Shortcuts Option-B

In comparison with the previous architectures, the 48-layer ResNet achieves 5% higher accuracy than the Highway network and 25% better performance than the equivalent plain net. On another note, it is visible from figure 8, that the network greatly overfits after epoch 12 since the training error still declines while in validation time it increases. That means that we should have used

a higher regularization penalty or even a dropout layer. The proposed architecture in [4] does not consist of regularization layers, such as dropout or maxout, however, the authors suggest that it might further improve accuracy. Ultimately, the residual network architecture with skip connections and identity shortcuts helped growing the network deeper, alleviate the degradation effect and achieve far better accuracy than the architectures presented in the previous sections.

3.3 Densely Connected Convolutional Networks

Densely connected convolutional networks or DenseNets [12] is the most recent (December 2016) *state of the art* architecture that exploits the depth of convolutional neural networks. The DenseNet architecture is based on the Highway and ResNet architectures which are using skip connections but it is taking this idea to the extreme. Figure 9 outlines the general idea which utilizes connectivity between the layers. Unlike ResNets where each layer is connected with identity mappings to the next layer, DenseNets connects each layer to every upcoming one. The reason behind this structure is that, if skip connections between each layer improves performance, creating a direct information route between every layer would achieve even better results. In addition, the authors in [12] argue that there is a great amount of redundancy in deep residual networks and performance could be improved by dropping layers randomly during training which is the basic inspiration for the current architecture.

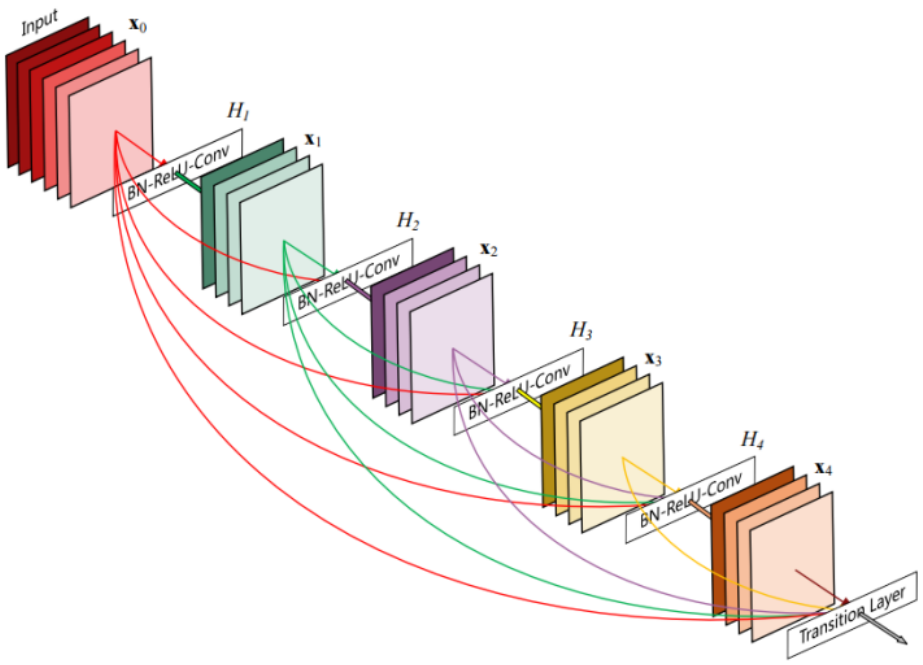

Figure 9: Densely connected neural network architecture. **Source**: [12]

Instead of residual blocks of the ResNet architecture (figure 7), the DenseNet architecture maximizes information flow by directly connecting a layer with all subsequent ones. This is realized by creating stacks of dense blocks where each layer is directly connected with all the previous layers in the current stack. In other words, the n^{th} layer in the stack receives the feature maps of all the preceding layers $x_0, \ldots, x_{n-1}$ as input,

$$x_n = H_n([x_0, x_1, \ldots, x_{n-1}]) \tag{2}$$

where H_n is the activation function of the n^{th} layer and $[x_0, x_1, \ldots, x_{n-1}]$ is the feature maps concatenation of the previous $n-1$ layers.

3.3.1 Architecture and Outline of Experiments

The implementation of DenseNet follows the strategy of the previous architectures. Each convolutional layer has the same feature map size across each stack. In order to preserve the same complexity in each stack, the layers have the same number of filters and when the feature map is halved, the number of filters is doubled. In contrast to the residual and highway networks, the downsampling is realized by 2x2 average pooling layers of stride 2 [12]. This was implemented with tf.nn.avg_pool function in tensorflow. Stacks of *6n* 3x3 convolutional layers with filters of sizes [32,16,8] are used respectively with *2n* layers for each filter size as table 9 shows.

24 Layers	34 layers	48 Layers
input 32 x 32 RGB image		
conv3-16		
conv3-16 conv3-16 } $x3$	conv3-16 conv3-16 } $x5$	conv3-16 conv3-16 } $x7$
2x2 average pool, stride 2x2		
conv3-32 conv3-32 } $x3$	conv3-32 conv3-32 } $x5$	conv3-32 conv3-32 } $x7$
2x2 average pool, stride 2x2		
conv3-64 conv3-64 } $x3$	conv3-64 conv3-64 } $x5$	conv3-64 conv3-64 } $x7$
conv3-10/100		
global average pooling		
Soft-max Output		

Table 9: DenseNet configuration. The convolutional layer parameter are denoted as "conv<receptive field size>-<number of channels>".

The aforementioned dense connectivity in each stack of convolutional layers is implemented by the concatenation of the previous outputs. In other words, each convolutional layer takes as input the output of each previous convolutional layer in concatenated form which is implemented by tf.concat function in tensorflow. A list structure is used to save the output of each layer so that it could be used as input in the deeper layers in the stack and every new output is added to that list. Each convolutional layer is implemented as in previous architectures with slim.conv2d function and the weights were initialized by the scaling variance initialization to help convergence [8]. The output of each convolutional layer is centered and normalized with batch normalization (tf.nn.batch_norm) [13]. The non-linear activation function for every layer is ReLu and in contrast to the residual network, the authors in [12] suggest a dropout layer (tf.nn.dropout) for regularization purposes with drop rate equal to 0.2. The network is further regularized with L2 regularization and weight penalty equal to 0.0001.

We experiment over the depth of the DenseNet with *n=[3,5,7]*, leading to 24, 34 and 48-layers networks so we could compare with the exact deep models of the previous architectures. The models are trained with minibatch size of 50 and momentum optimizer with learning rate 0.1 and momentum 0.9 where the learning rate is divided by 10 every 10 epochs to help the optimizer converge. We track the error and accuracy for the training and validation sets which were implemented as in section 2.1 respectively. The experiments were conducted for 30 epochs.

3.3.2 Results and Discussion

The accuracy results of table 10 indicate that the DenseNet architecture is capable of alleviating the degradation phenomenon of deep convolutional networks. The 48-layer network achieves the highest accuracy of 70.08% and 40.1% for both datasets. The dense connectivity of the current architecture confirms the hypothesis that information flow between every layer in the stack could achieve better performance than the skip connections of the residual network. The difference in performance is not particularly high but yet substantial between 1-2% for each model. This is probably due to the fact that the densenet is more regularized and does not overfit throughout the experiments. Figure 10 confirms this since the training and validation error curves both decline apromaximately with the same pace and continue to decline even until the end of the experiment.

Nr. of Hidden Layers	Accuracy CIFAR-10 (%)	Accuracy CIFAR-100 (%)
24	70.01	39.76
34	70.03	40.03
48	70.08	40.1

Table 10: Validation Accuracy of the ResNet for CIFAR-10/100 over depth for two shortcut options A & B.

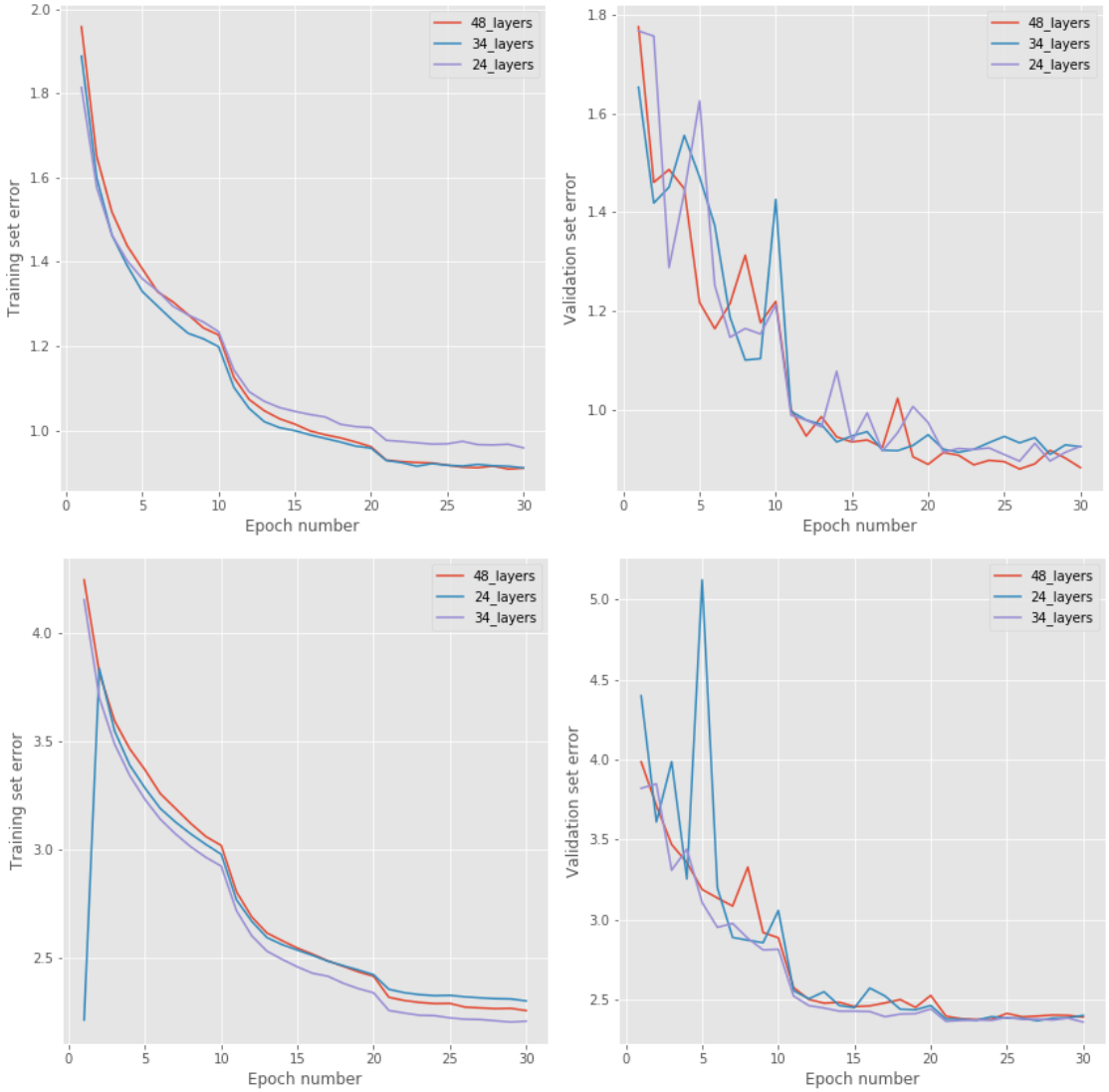

Figure 10: Training and validation error figures for the DenseNet. **Top**: CIFAR-10, **Bottom**: CIFAR-100

4 Conclusion

The purpose of this paper was to identify and prove the accuracy degradation phenomenon in deep convolutional networks used for image recognition tasks, as well as, to propose techniques and architectures that could alleviate it and help the networks to grow deeper. We were highly motivated by the fact that deep convolutional neural networks could create better feature representations of the input image. We argue that deeper models achieve worst performance due to the degradation phenomenon, which happens because of the highly complexity of deep networks and the difficulty of the SGD solvers to optimize them and not by the vanishing of the gradients nor by overfitting [5, 7, 9, 11, 15]. In section 2, we experimented with popular convolutional neural networks and the baseline feedforward architecture of the previous coursework and proved that the accuracy saturates when the networks grow deeper because of the increased complexity (degradation). In particular, as the networks grow deeper their error is higher and at the same time they achieve less accuracy. Table 11 shows that the accuracy of the feedforward, VGG [3] and plain networks [4] degrades while they grow deeper.

Model	Accuracy CIFAR-10 (%)	Accuracy CIFAR-100 (%)
FC-1	44.6	21.3
FC-3	**52.9**	**28.1**
FC-5	52.4	26.8
FC-8	51.4	23.8
FC-14	47.8	17.9
VGG-8	**63**	**32**
VGG-16	61.25	28.8
VGG-20	48.9	22.7
PlainNet-24	**64.7**	**31.6**
PlainNet-34	55.4	23.9
PlainNet-48	45.5	20.4
Highway-24	**67.7**	39.2
Highway-34	65.4	**39.9**
Highway-48	65	30.07
ResNet-A-24	68.4	39.9
ResNet-A-34	69.3	40
ResNet-A-48	68.9	40.1
ResNet-B-24	68.7	39.9
ResNet-B-34	70.1	39.9
ResNet-B-48	**70.2**	**40.1**
DenseNet-24	70.01	39.76
DenseNet-34	70.03	40.03
DenseNet-48	**70.08**	**40.1**

Table 11: Overall Validation Accuracy for CIFAR-10 and CIFAR-100.

In section 3, we presented three *state of the art* deep convolutional neural network architectures which effectively coped with the degradation phenomenon and could achieve improved performance with hundreds of layers. All of them are based on the concept of passing information directly between layers. The highway network passes information through gates which helped achieve higher accuracy in deeper models but still showed some signs of degradation on CIFAR-100 [17]. The residual network architecture [4], currently the most popular architecture for image recognition tasks, efficiently alleviated the degradation effect. By taking advantage of skip connections between layers, it achieved high accuracy with the deepest 48-layer model which we experimented. Finally, we experimented with a fairly new architecture which densely connects convolutional layers [12], based on the same idea of residual networks to help information flow directly between layers. The DenseNet achieved the higher accuracy throughout our experiments for both datasets as table 11 presents. We assume that this happened due to the high dense connectivity that the architecture proposes and the better regularization that was used.

4.1 Further Work

Throughout the paper, the experiments were conducted for networks up to 48 layers. This experimental choice and constraint was due computational limitations. As further work, we could experiment with deeper models to hundreds and even thousand layers and show that the proposed architectures could overcome the degradation obstacle in this context [4, 12, 17]. In addition, we could also experiment with fractal networks. This architecture is proposed to help convolutional neural networks achieve better performance with ultra deep models. Their structural layout is precisely a truncated fractal and contains sub-paths of different length. The information signal is transformed by a filter and non-linear activation before being passed to subsequent layers which do not include any direct connections between them [18].

References

[1] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. *Imagenet classification with deep convolutional neural networks*. Advances in neural information processing systems. 2012

[2] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). *Dropout: a simple way to prevent neural networks from overfitting*. Journal of Machine Learning Research, 15(1), 1929-1958.

[3] Simonyan, Karen, and Andrew Zisserman. *Very deep convolutional networks for large-scale image recognition*. arXiv preprint arXiv:1409.1556 (2014).

[4] He, K., Zhang, X., Ren, S., & Sun, J. (2016). *Deep residual learning for image recognition*. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 770-778).

[5] Bengio, Yoshua, Patrice Simard, and Paolo Frasconi. *Learning long-term dependencies with gradient descent is difficult*. IEEE transactions on neural networks 5.2 (1994): 157-166.

[6] Ciresan, Dan Claudiu, et al. *Convolutional neural network committees for handwritten character classification*. Document Analysis and Recognition (ICDAR), 2011 International Conference on. IEEE, 2011.

[7] Glorot, Xavier, and Yoshua Bengio. *Understanding the difficulty of training deep feedforward neural networks*. Aistats. Vol. 9. 2010.

[8] Mishkin, Dmytro, and Jiri Matas. *All you need is a good init*. arXiv preprint arXiv:1511.06422 (2015).

[9] LeCun, Yann A., et al. *Efficient backprop*. Neural networks: Tricks of the trade. Springer Berlin Heidelberg, 2012. 9-48.

[10] Saxe, Andrew M., James L. McClelland, and Surya Ganguli. *Exact solutions to the nonlinear dynamics of learning in deep linear neural networks*. arXiv preprint arXiv:1312.6120 (2013).

[11] He, Kaiming, et al. *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*. Proceedings of the IEEE international conference on computer vision. 2015.

[12] Huang, G., Liu, Z., Weinberger, K. Q., & van der Maaten, L. (2016). *Densely connected convolutional networks*. arXiv preprint arXiv:1608.06993.

[13] Ioffe, Sergey, and Christian Szegedy. *Batch normalization: Accelerating deep network training by reducing internal covariate shift*. arXiv preprint arXiv:1502.03167 (2015).

[14] Sermanet, Pierre, et al. *Overfeat: Integrated recognition, localization and detection using convolutional networks*. arXiv preprint arXiv:1312.6229 (2013).

[15] Szegedy, Christian, et al. *Going deeper with convolutions*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.

[16] He, Kaiming, and Jian Sun. *Convolutional neural networks at constrained time cost.* Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.

[17] Srivastava, Rupesh Kumar, Klaus Greff, and Jürgen Schmidhuber. *Highway networks.* arXiv preprint arXiv:1505.00387 (2015).

[18] Larsson, Gustav, Michael Maire, and Gregory Shakhnarovich. *Fractalnet: Ultra-deep neural networks without residuals.* arXiv preprint arXiv:1605.07648 (2016).

[19] Hochreiter, Sepp, and Jürgen Schmidhuber. *Long short-term memory.* Neural computation 9.8 (1997): 1735-1780.