TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

MIKKO SAARI

THE EFFECT OF TWO HYPERPARAMETERS IN THE LEARNING PERFORMANCE OF THE CONVOLUTIONAL NEURAL NETWORKS

Bachelor of Science

Examiner: Professor Heikki Huttunen

ABSTRACT

MIKKO SAARI: The effect of two hyperparameters in the learning performance of the Convolutional Neural Networks
Tampere University of Technology

Bachelor of Science Thesis, 26 pages, 1 Appendix page
June 2018
Bachelor's Degree Programme in Science and Engineering
Major: Machine Learning
Examiner: Professor Heikki Huttunen

Keywords: Deep Learning, Machine Learning, Convolutional Neural Networks, Image Recognition

The research topic of this work was to study the effect of two special hyperparameters to the learning performance of Convolutional Neural Networks (CNN) in the Cifar-10 image recognition problem.

The first hyperparameter, that was chosen to be studied, was the depth of the CNN, which is known to effect to the amount of the feature extraction. The second hyperparameter, was the use of the special regularization technique called Dataset Augmentation (DA), which is known to increase the amount of training data available artificially so that there is got extracted and taught more features from the members of each classes to the networks.

The hypothesis was that the increment of the depth and the training data would improve the learning accuracy, especially by improving the testing accuracy, which measures the ability of generalization of learned features for the model.

The work was implemented by the high-level Deep Learning software called Keras, of which source code is freely available on Github. For the CNN needed in the work there was found ready Python implementation, which was modified slightly by adding a few code lines.

The depth of the networks was incremented first without Dataset Augmentation, by adding the number of the convolutional layers one-by-one from four to eight. The same was repeated while the augmentation was set on.

The results were partly equivalent, partly contradictory with the hypothesis. The increment of the depth increased both the training and testing accuracy, when the Dataset Augmentation was not used. Instead if it was used, while the training accuracy still improved, the testing accuracy dropped, even below the accuracy achieved by the original networks.

The natural conclusion is that both increasing of the depth and the amount of data artificially effect the overfitting. Instead to apply both at same time, only either of them should be used.

TIIVISTELMÄ

MIKKO SAARI: Kahden hyperparametrin vaikutus konvoluutioneuroverkon oppimiskykyyn
Tampereen teknillinen yliopisto
Kandityö, 26 sivua, 1 liitesivu
Kesäkuu 2018
Tietotekniikan kandidaatin tutkinto-ohjelma
Pääaine: Koneoppiminen
Tarkastaja: professori Heikki Huttunen

Avainsanat: syväoppiminen, koneoppiminen, konvoluutioneuroverkot,kuvantunnistus

Tämän työn tutkimusaihe oli tutkia kahden eri hyperparametrin vaikutusta konvoluutioneuroverkon oppimiskykyyn Cifar-10 kuvantunnistusongelmassa.

Ensimmäinen tutkittavaksi valittu hyperparametri oli verkon syvyys, jonka tiedetään vaikuttavan piirreirroituksen määrään. Toinen oli Datajoukon augmentoinniksi kutsuttu regularisointimenetelmä, jonka tiedetään lisäävän käytettävissä olevan harjoitusdatan määrää keinotekoisesti siten, että kunkin luokan edustajasta saadaan irroitettua ja opetettua enemmän verkolle piirteitä.

Hypoteesi oli, että syvyyden lisääminen ja datan määrän kasvattinen lisäisivät oppimiskykyä, erityisesti parantamalla testitarkkuutta, joka mittaa mallin kykyä yleistää oppimansa piirteet.

Työ toteutettiin korkean tason syväoppimisohjelmistolla nimeltä Keras, jonka lähdekoodi on vapaasti saatavilla Githubista. Tarvittavalle CNN:lle löytyi valmis Python-toteutus, jota muokattiin kevyesti lisäämällä muutamia koodirivejä.

Verkon syvyyttä lisättiin ensin ilman että Datajoukon augmentaatiota olisi käytetty, lisäämällä yksi kerrallaan konvoluutiotasojen määrää neljästä kahdeksaan. Sama toistettiin kun augmentaatio oli asetettu päälle.

Tulokset olivat osittain yhtäpitäviä, osittain ristiriitaisia hypoteesin kanssa. Verkon syvyyden lisääminen kasvatti sekä harjoitus- että testitarkkuutta, kun Datan augmentointia ei käytetty. Sen sijaan jos sitä käytettiin, harjoitustarkkuuden entisestään parantuessa testitarkkuus putosi, jopa alle tarkkuuden, joka saavutettiin alkuperäisellä verkolla.

Luonnollinen johtopäätös on, että sekä syvyyden kasvattaminen että datan määrän lisääminen keinotekoisesti aiheuttavat ylioppimista. Sen sijaan että molempia käytettäisiin samanaikaisesti, vain jompaa kumpaa tulisi käyttää.

PREFACE

This Bachelor's work was done during the Spring 2018 for the Bachelor's Seminar in Signal Processing in Tampere University of Technology. Since my interest on the Deep Learning, I was very motivated to work on the topic. This work was extremely interesting since it introduced to the Deep Learning which has become one of the hottest topic in the Computer Science. Especially I want to thank professor Heikki Huttunen and professor Joni Kämäräinen, who both gave extremely valuable advice during the writing process and examined this work, and also my friends and family for their support.

In Tampere, Finland, on 22 June 2018

Mikko Saari

CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

CNN	Convolutional Neural Networks
MLP	Multilayer Perceptron
SGD	Stochastic Gradient Descent
BP	Backpropagation
DA	Dataset Augmentation
GPU	Graphical Processing Unit
Relu	Rectified Linear Unit

L	loss function
$\mathbf{F}$	classification function
$\mathbf{w}$	weight parameter
$\mathbf{x}$	input data sample
$\mathbf{y}$	output class label

1. INTRODUCTION

The image recognition is widely used in different applications, like Computer Vision and face recognition and is a crucial part of many real intelligent autonomous systems.

The main challenge is that images of real-world objects are typically complex and high-dimensional, having complicated relations between its smallest elements, pixels.

Thus it is extremely difficult to construct any rule that would recognize the image directly by comparing the values of the single pixels.

The better approach is first detect abstract properties, called features, that separate the images from different categories mostly and then to formulate the decision rule by help of them.

Deep Learning allows the program to build autonomously a deep graphical representation of the data where the abstract concepts are formed by combining the simpler ones. The representation of the data is in a way scaled depending on the amount of data that is offered and the application for which it is needed.

Especially the most applied architecture for the image recognition, Convolutional Neural Networks, detects the features and by help of these forms the most suitable probabilistic classification rule.

It is able to find the best filters to extract features to offer the optimal classification accuracy.

Its architecture is designed to exploit the special properties encountered in the images such as high local correlations between nearby pixels , and invariances under geometrical transformations, like translation or rotation. Inspite of the benefits of DNNs they were not applied before in a big amount, since the computationally efficient implementations were missing, and it was assumed that they are too eager to overfit and hard to be regularized.

The fast development of Graphical Processing Units (GPU) made possible to build the parallel algorithms for CNNs and when the Canadian research group made succesfull experiments with the CNN, the use of Deep Learning exploded.

After that it has been applied succesfully to solve many kind of Machine Learning tasks, especially those having complex data, like in the image and speech recognition.

In this work it is studied, how the modifications on the two hyperparameters, the increment of the depth of the CNN and the use of the regularization technique called Dataset Augmentation, would effect on the learning performance of CNN. The hypothesis was the use of both would improve it, but the results of experiments were contradictory.

The theoretical background of the principles of the Deep Learning needed to understand the research topic are given in chapter 2.

The implementation and results are described in the chapter 3, and the conclusions in the chapter 4.

2. THEORETICAL BACKGROUND

In this chapter the theoretical background is explained that is important for understanding the research topic of this work.

The chapter is divided to two sections, 2.1 discusses generally on the Deep Learning and 2.2 describes the structure of CNN and regularization techniques especially applied for it.

2.1 Deep Learning

2.1.1 Machine Learning

The traditional way to solve problems algorithmically is to describe instructions in details, which needs that the relations between the elements of the data are known explicitly. Therefore this approach has been enormously hard for the problems, in which these relations are complex, and often they still are for people intuitive to solve, like image recognition. [4,p.2]

In Machine Learning, the essential properties of the data and their representation is learned by program itself, by allowing it to improve its performance in a learning task by help of the experience.

It can be formally defined as a tuple

$$(T, P, E) \tag{2.1}$$

,where

$$T$$

means a given learning task,

$$P$$

is a performance measure, describing how succesfully the program solves the task, and

$$E$$

is an experience, by which the performance is improved.

[6,p.2]

Learning can be divided to the supervised, reinforcement and unsupervised mode.

Their main differences are in an amount and a type of the experience that is available for the program to be used for learning. Here only the first mode is discussed, since the CIFAR-10 image recognition task is supervised.

The essential part of the learning is to improve the performance of the program in solving the task.

It is measured usually in the supervised tasks by help of an objective function called loss or cost, that describes how much the prediction of the model produced by the algorithm differs from the target output.

In this case there exists the quantitative feedback from each of the training sample and the error is usually averaged over the whole training data.

The ultimate goal would though be to minimize the loss for the new unseen data samples, which is assumed to obey the distribution that would generate arbitrary samples of the data correctly.

Unfortunately the learning model can be tested only by the limited amount of the testing data. [4,p.272]

Supervised tasks can be often represented as a function approximation problem, that can be divided to regression and classification tasks. Their main difference is that in the regression the target values are real-valued whereas in the classification they are discrete set of class labels. Thus in the classification dimension of the target output is typically much smaller than the dimension of the input. [9,p.3]

In Cifar-10-image recognition the task is to classify an image under correct class label. Practically in the image recognition the algorithm should be able to answer the question ”what is the class label of the given image”.

The learning algorithm to solve the task can be seen consisting of a model, an optimizing algorithm and a dataset.

[4,p.97]

In this work a learning model is CNN, the optimizing algorithm is Stochastic Gradient Descent and a dataset is Cifar-10.

In the classification task the pairs of the dataset can be seen as matrix:

$$\mathbf{X} \tag{2.2}$$

being the matrix of input training samples and

$$\mathbf{y} \tag{2.3}$$

is a target output class label to which the given image belongs.

For instance in the Cifar-10 the input is a 32x32 pixel image and the output its class label, describing to which of 10 available class categories it belongs.

Cifar consists of images of the animals and vehicles, each of classes being mutually exclusive to each other, so that each image belongs exactly and only to one of them. [13] To test the generalization ability Cifar-10-dataset is divided to two parts.

The bigger part, of 50.000 images, forms training data, by which the model is trained and smaller, 10.000 images, a testing set, by which the model is tested.

Formally, the classification can be represented as a function:

$$\mathbf{F} : \mathbf{x} - > \mathbf{y} \tag{2.4}$$

,where **F** is a classification function.

Before the classification rule between images and categories can be found, the images should be preprocessed by reducing the dimensionality of the data and the amount of information so that only the essential properties of data are preserved. They are called features, describing how images differ from each other.

In the feature extraction, the original data is mapped to the feature space.

This is called as the feature extraction, traditionally it has formed the most time-consuming part in the design of the Machine Learning algorithm.

The Deep Learning, like CNN, overcome this by allowing the algorithm itself to find the most optimal feature extraction at same time as learning the best classification rule. [4, p.3]

Thus in the high level of abstraction, the learning process is a compression of the original data to the form from which the learning task can be most optimally solved.

2.1.2 Deep Neural Networks

The Deep Neural Networks (DNN) is a learning model, originally inspired by the research on the biological nervous system, and still sometimes taking inspiration on it.[3, p.23]

Nowadays they are more seen as the non-linear generalization of the linear models, seeking an optimal function approximation between the input and the output of data samples.

The most basic type of DNN is a feedforward, densely-connected Multilayer Perceptron (MLP), without recurrent connections.

It can be seen as an acyclic graph, where the values of verteces are computed when the data is fed through the networks and the values of edges corrected when the parameters are adapted. [6,p.83]

The simple linear models like LDA and logistic regression are computably efficient, still without overfitting easily, and therefore attractive to be applied in the classification tasks.

[1, p.261]

They don't have enough capacity to learn the features from the complex high-dimensional data like real world images and non-linear decision boundaries, that would form non-convex sets in the feature space.

To overcome this there is introduced the non-linear depedence between the approximated function and the weight parameters. At least one non-linear mapping is applied for the feature map, and thus more interesting features can be found than by linear models.

In fact, the universal approximation theory tells that MLP with a single nonlinear mapping can approximate any Borel-measurable, real-valued function. The challenge is to find the parameters, and in the way that is computationally effective and accurate.

The approximation is achieved already by using single hidden layer, and the introduction of new layers helps to find new, more abstract features by transforming the data to new feature spaces. Thus the hidden layers can be applied in special tasks like for finding the new features, or reducing the noise or amount of the information, like done in Pooling layers, or to implement the classification step itself, like Softmax.

The main problem that has discouraged applying multiple hidden layers, has been eagerness of DNN to overlearn and lack of computationally efficient implementations. Since DNN is naturally distributed, parallel computing model, the sequential CPU is not optimal for them. For this reason the fast development of GPU computing yet has made possible to apply Deep Learning.

In the supervised learning task, the training experience consists of input and output pairs, and the DNN seeks the function, that would approximate the functional depedence between them optimally, still mainly reducing the generalization error instead the training error.

It is implemented by decomposing the approximation function to the combination of the simpler functions, where the successor function is defined in terms of the predecessor function. Thus there is constructed inference from the input to the output.

Each hidden layer defines a new function thus offering new abstraction level for the data, and they are chained in linearly, layers mapping the data so forward towards the output layer, that in the output layer the values would be as close as possible to the desired values.

After that the difference between the signal of the output layer and the known target is computed, and from this information the value of the cost function is evaluated.

For instance in the CNN the first layers detect rough, low-level features and the deeper levels more abstract ones by combining the simpler ones.

The depth is usually defined as a number of layers, which have a nonlinear activation function, so that the depth is one layer more than the number of hidden layers.

Usually the Deep models are defined to have at least two hidden layers, in modern architectures much deeper networks, even consisting of hundreds of hidden layers, are used.

Naturally more layers are applied, more computations and longer training time is needed.

The basic unit of MLP is a perceptron, called also neuron, where the linear discriminant takes as an input the input signal, multiplies every input value with the corresponding weight parameter and sum together the weighted inputs.

Usually for each weighted sum is added a non-zero constant, bias term, thus making the feature mapping as an affine mapping. [3, p.34]

This can be seen equivalently as a dot product between the input vector and the weight parameter vector.

Then the nonlinear activation function maps the whole feature vector, forming an activation map. It is fed as an input value for the neurons in next layer.

It can be thought that every weight describe how much single feature effects to the prediction output.

A single perceptron implements one discriminant function, thus returning single real value as an output.

The perceptron are usually set to parallel so that they can find for same input vector many feature values, thus producing the feature vector for single sample instead single scalar value .

The linear part of the discriminant function can be represented as:

$$net_j = \mathbf{w_j} \cdot \mathbf{x} + b = \sum_{i=1}^{M} w_{ji} x_i + b \tag{2.5}$$

where

$$b$$

, is a bias term or threshold value. [6, p.86] The linear feature map is mapped by a nonlinear activation, thus the depedence of the output on the parameters becomes nonlinear, and feature values can be more suitably represented.

$$y_j = f(net_j) \tag{2.6}$$

,where

$$f$$

is an activation function.

The most used activation in the literature is a logistic sigmoid, which squashes the linear output to the real number between 0 and 1. It can be considered for example as a probability that the feature effects to the output.

It in a way decides which features from the feature map detected by linear mapping are important and which are not. For instance the logistic sigmoid which returns real value between 0 and 1, maps the big significant values close to 1 and less important to 0. Relu in other hand preserves the feature map as it is for non-negative values and forgets other. [4,p.170]

For MLP the problem is also that it does not scale well to the images, because the number of parameters grows exponentially, depending on the dimension of the image. [2]

One can say that general MLP takes too much details in account, that are not essential for the image recognition, but instead even disturb solving it.

Instead the CNN is designed to take those most likely essential features for the image classification, by exploiting the properties that are typically encountered in them.

For other kind of data and general tasks, MLP can be more accurate, but it is obvious that for the image recognition CNN is more suitable choice, unless the images are very simple like in MNIST-recognition task.

The structure of CNN is more discussed in next section.

2.1.3 Training

The crucial part of the learning is the improvement of the performance by help of the experience. This is expressed in the supervised learning by forming the objective function that is computed from the losses of the single training samples.

It is called as a loss, which describes the differences between the target and the actual output the model predicts.

The value of loss is computed for every single training sample but the total value is averaged over the training set.

The target is that such a model would be found that covers not only the samples of limited training samples but also totally new, unseen ones.

The optimization of the cost function directly by minimizing the training error of samples is not thus the fundamental goal.

Instead the task is to minimize the generalization error of samples obeying the data-generating distribution, indirectly by help of the training error of samples obeying the empirical distribution.

[4,p.272]

Often there is added a regularization term, and thus the minimum point of the loss is changed to help to generalize the model better.

For this reason the training is stopped already when the generalization error starts to grow, instead to finding the point where the loss would be minimized.

In the image recognition the most appropriate cost is the cross entropy, optional form of the maximum likelihood, since it tries directly to minimize the distance between the probability distribution the model predicts and the target one.

It can be expressed as:

$$L(w) = -\sum_{i=1}^{C} y_i log(y_i) + (1 - y_i) log(1 - y_i) \tag{2.7}$$

$$y_i$$

is a class label of i. class and

$$C$$

is a number of class labels in data set [6, p.118]

When the samples are correctly classified the cross-entropy achieves its minimum value.

Practically for the image classification it means when the objective function is optimized, the model most probably predicts correctly under which class label the image belongs to. It is minimized when the samples drawn from each classes are belonging to correct ones, and the actual output probability the CNN produces is as close as possible to the target distribution.

The main challenge encountered in MLP is that desired output is directly shown only for the output layer, but the intermediate layers do not see it directly, so that they could produce the desired values of output layer.

[4,p.165]

The learning algorithm should be able to decide how to apply the hidden layers to approximate the target.

This is called as a credit-assignment problem, meaning the optimal values of the intermediate steps of algorithm to achieve the final output are not seen directly but should be deduced indirectly, by using the experience.

[4,p.272]

The problem is thus to learn both the discriminant function and the nonlinear mapping at same time. Because of nonlinearity the closed form solution like LMS does not exist, but instead the optimal parameters are sought by some iterative optimization algorithm.

Training consists of the feeding and adaptation phases, in feeding the training samples are fed through the networks, and the value of the cost function for them is computed.

After that in the adaption the error-correction values for parameters are computed to decrease the loss for each layers between the input and output, progressing backwards from the final layer to the first one.

Usually the data is fed in batches, so that after feeding a subset of training data, the corresponding changes for the weights are computed and then backpropagated.

For the Deep Learning the adaption is implemented by Stochastic Gradient Descent (SGD) and Backpropagation (BP). The SGD seeks the minimum point for the loss function, and BP finds the paramers of minimum point for each layers.

The principle idea behind the SGD is that the negative gradient of the cost function tells the direction where the function decreases, by pointing towards the nearest local minimum of the error surface.

Thus by following the path to which it directs stepwise, the minimum point of the function should be found.

Since it is known the change of the parameters is directly proportional to the gradient of the cost function, the amount of the correction for the weights can be expressed generally:

$$\Delta w = \mu \nabla L(w) \tag{2.8}$$

,where

$$\mu$$

is a positive real-valued constant, called step size or learning rate, that scale the corrections of the parameters. If it is chosen to the suitable value, it makes convergence faster. Instead if it is too big, the algorithm can diverge, and if too small, the convergence can be extremely slow. The SGD is sensitive for the initial values of the model, due to the nonlinear depedence between the input and the loss function, and the final result of the optimization can change dramatically depedending on their initial values.

$$\nabla L(w) \tag{2.9}$$

is a gradient of the loss function, consisting of all the partial derivatives of the function in relation to its weight parameters.

There exist several options of learning algorithms, most being variants of SGD, which apply heuristical methods to make the convergence more reliable and fast.

The basic version SGD for function optimization can be described in a following way:

Initialize weights randomly to small value, close to zero. Set threshold, so that if the value of gradient is smaller, the training is terminated, since the minimum point is most likely encountered.

$$w_0 = rand$$

As long as the gradient is bigger than the threshold, the next procedure is iterated:

$$\begin{aligned} while \nabla L(w_k) &< \theta \\ w_{k+1} &= w_k + \Delta w_k \\ &= w_k + \mu \nabla L(w_k) \\ &= w_k + \mu \frac{\partial L(w_k)}{\partial w_k} \end{aligned}$$

,where k is a number of number iterations of SGD.

$$\theta$$

is threshold value. If the value of the gradient decreases below it, the training is terminated.

[6, p.91]

In the basic gradient descent rule whole training data set is fed and yet after that the parameters adapted, but in SGD the data is fed in randomly chosen subsets, and the adaptations are done for each of them.

The actual values for the correction of the weights for each layer are computed by the Backpropagation , which is most applied error-correction mechanism in Deep Learning. It makes possible to tell to the hidden layers such corrections for the parameters that would decrease the loss.

Feedforward propagation:

1. Feed the batch of training data to the networks propagating it through the networks and compute the corresponding losses.

Backward propagation:

The error correction terms are computed recursively backward, first between the final hidden and the output layer, then to the second last progressing towards the first layer.

2. Compute error term for weights between every hidden and output units .

$$\delta_j = -(t_j - o_j)y_j(1 - y_j) \tag{2.10}$$

3. For each output unit compute its corresponding error term, expressing it by help of the previously computed correction term

$$\delta_i = \sum_{j=1}^{C} \delta_j w_{ji} \frac{\partial y_i}{\partial net_i} \tag{2.11}$$

4. Update each network weight.

$$w_{ji} = w_{ji} + \Delta w_{ji} \tag{2.12}$$

where

$$\Delta w_{ji} = \mu \delta_j x_i \tag{2.13}$$

[6,p.93]

The gradient of loss function is computed so that all the partial derivatives in the relation to the weight parameters are computed, first for the weights between hidden and output layer.

By the chain rule of differential calculus the partial derivative of the loss is decomposed to two parts.

$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial L}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} = \delta_j x_i \tag{2.14}$$

[6,p.102]

The error-correction term of the loss function is computed for

Output-hidden layer:

$$\delta_j = \frac{\partial L}{\partial net_j} = \frac{\partial L}{\partial y_j} \frac{\partial y_j}{\partial net_j} = -(t_j - y_j) \frac{\partial f(net_j)}{\partial net_j} = -(t_j - y_j)y_j(1 - y_j) \tag{2.15}$$

[6,p.102] , where activation function

$$f(net_j) = \frac{1}{1 + \exp(-net_j)} \tag{2.16}$$

is the logistic sigmoid, most commonly applied activation function in the literature, due to its analytical properties and the way it resembles the transfer function of the biological neurons.

The formula simplifies even more if there is applied Relu, since it has constant derivative, 1 for non-negative input arguments, and 0 otherwise.

In similar fashion for hidden-input

$$\begin{aligned}\delta_i = \frac{\partial L}{\partial net_i} &= \sum_{j=1}^{C} \frac{\partial L}{\partial net_j} \frac{\partial net_j}{\partial net_i} \\ = \sum_{j=1}^{C} \delta_j \frac{\partial net_j}{\partial y_j} \frac{\partial y_j}{\partial net_i} &= \sum_{j=1}^{C} \delta_j w_{ji} \frac{\partial y_j}{\partial net_i} \\ &= \sum_{j=1}^{C} \frac{\partial y_j}{\partial net_j} - (t_j - y_j) w_{ji} \frac{\partial y_j}{\partial net_i}\end{aligned}$$

[6,p.103] where

$$\delta_j$$

is error-correction term computed previously for output-hidden layers

Fortunately it is not essential to find the exact global minima, but such one that suits the need of the learning task offering the performance high enough.

SGD and BP can be trapped in the local minima since they only seek the point where the converging constraint holds, therefore the error can be reasonably high.

If early stopping is enabled, SGD finds more likely the point that would offer good generalization accuracy.

Actually still more parameters are used, more unlikely is that the point algorithm finds would be non-suitable local minimum.

Thus for CNN which has a few million parameters, the minima found by SGD and BP are most likely those that are sought or at least suitable enough for the purpose of the application.

2.2 CNN

2.2.1 Structure of CNN

The CNN are a special type of feedforward neural networks that are designed to the image recognition, by incorporating two important constraints to their architecture.

Thus they take better account the special statistical and topological properties of the image, than the general Multilayer Perceptron.

First of all it is known that nearby pixels in the image have more likely strong correlations than pixels farther each other. [9,p.267]

The MLP has a parametrization between each input and output value, and thus the number of free parameters is huge.

In MLP every part of the image are parametrized by different weight values, but in CNN due to the sparse connections and the parameter sharing same parametrization are applied for different regions in the image.

As a consequence almost equivariant representation for the image is found, meaning the same image is recognized though it would be shifted or rotated. It seems clear that in the real-world applications like in robotics this property is necessary.

The CNN seeks the probabilistic mapping from the images of data set to the class labels, by first transforming them to the set of features and then mapping features to the categories.

The theoretical optimal performance of the CNN differs only a bit from MLP and is much more faster achieved. [5, p.1]

The CNN have more hyperparameters than MLP due to the modifications, for example the size of the convolutional kernel, the breadth, which here is three for RGB color channels and use of zero padding.

In zero padding the edge of image is filled artificially by zeros so that also the corners can be convolved for the output, even they before the padding don't have enough pixels.

Generally the convolution function in mathematics and signal processing is defined between two real-valued function. It can be used to model for example the effect of the measurement error or noise into the observed signal.

$$\int_{-\infty}^{\infty} w(T)x(t-T)dT \tag{2.17}$$

,where

$$w(T)$$

is noise function and

$$x(t)$$

is a pure uncorrupted signal. Often the weight is restricted to the range between 0 and 1, which can in a sense thought describing the probability of the single pixel to effect on the feature map.

$$T$$

parameter shift, i.e time delay.

The convolution used in the Convolutional Neural Networks is two-dimensional, and discrete. Instead the convolution would be computed over the whole signal the weights are defined to be non-zero only inside the limited area.

$$y(m,n) = \sum_{i=0}^{M} \sum_{j=0}^{N} w(m,i,n,j) x(m-i,n-j) \tag{2.18}$$

,where

$$w(m,i,n,j)$$

is the kernel or weight function by which the image is filtered. [4,p.328]

The size of kernel is chosen to be approximately at the same size as the feature that is sought.

The first convolutional layers find the rough, simple properties like the corners or edges on the image, and those applied later form the abstract features from the earlier found, that can be exploited in the classification. [12]

The sparse connections help to detect the local features, by restricting the area to be studied by single convolution to the subset of the pixels in the image.

The convolution operation is done for every pixels in every color channels, thus having three parallel convolution operation at same time, making it easy to be parallelized. By combining the three different colors, red, blue and green, there has been got any arbitrary color.

When seen as a form of graph, the adjacent neighbor layers are connected by sparsely, so that only a subset of the units of the predecessor layer have been connected to the single unit in the successor layer.

[11]

All the nodes in input layer are not connected to every of nodes in hidden layer, but because every of nodes in hidden layer have connection to the final output layer, every input nodes are effecting to every of output nodes.

The parameter sharing means the same parametrization of the weights is applied for each subset of the nodes. Thus the weights are restricted to take same values in the different convolution kernels in a single layer.

The other modifications is the weight sharing which means that same parameter values are applied during the single convolution operation. Because of sharing weights the same feature is detected in single convolutional layer through the whole image, i.e the edge or contour.

The second part of the Convolutional layer is the activation layer, that implements a nonlinear activation function, by which the nonlinear depedence on the weight parameters is achieved and thus the universal approximation property.

In theoretical deduction there is chosen for the activation, i.e logistic sigmoid because of analytical properties, but for practical model the most used is Rectified Linear Unit (Relu) because of its computational efficiency and accuracy.

In fact it is linear in both parts of its domain, when the input is limited to take either negative or non-negative values but non-linear if real-values from both side of zero are in the domain.

[12] It can be shown also kind of a classification already, where the non-negative inputs are classified to the category ”keep” and negative to ”ignore” by setting it zero.

This makes the actual evaluation of Relu easy, because it can be implemented as a conditional statement of branches where the value of function is preserved unchanged if it is non-negative and otherwise set to zero.

Instead for example sigmoid needs arithmetics by floating points, and thus more processing, as a consequence the probability of the computational noise and error increases.

Even if the Relu makes the computing in the feeding data more simple, in the adaptation phase there occurs a small theoretical problem.

Relu is continous but not differentiable when the input gets value zero, and it is shown in the graph which has sharp edge when the input is zero.

[4,p.170]

Theoretically this could prevent the SGD, and BP to work, but practically it can be omitted since the behavior of the function in a neighborhood of zero is known. When the input of Relu approaches to zero, the output value approaches too, and therefore whole difference quotient approaches to zero.

The Pooling is an operation where the feature map is subsampled, to reduce the dimensionality, variance of the data and the amount of the information.

After the feature detector implemented by convolution and activation layer there is the subsampling layer that reduces the noise of signal and unnecessary information and so rises essential features better to the stage in the feature map.

[4,p.338]

The goal is to delete part of the signal values, and still keep essential features invariant, still without loosing too much information.

The Pooling that is used in the CNN of example Cifar-10 code in Keras is MaxPooling which takes the largest value of the subregion in the activation map and forgets other values.

For that reason it cannot been applied too many times, since the dimension of feature map can drop too fastly, even to zero and thus it does not contain enough information for the purpose of the classification anymore.

The final layer of the CNN is Softmax, which is a generalized version of the logistic discrimination for multi-category classification.

It classifies the image under one of the class labels, by returning the probability for it to belong to the one class between 0 and 1.

Input is the feature vector representing the essential properties of the image that is output of the previous layer of CNN and output is the 10-dimensional vector representing discrete probability distribution.

It can be expressed as

$$\Pr\left(y_i \mid \mathbf{x}\right) = \frac{e^{\mathbf{W_i}\cdot\mathbf{x}+b}}{\sum_{j=1}^{C} e^{\mathbf{W_j}\cdot\mathbf{x}+b}} \tag{2.19}$$

[4,p.79] , where

$$c$$

is number of class labels

and

$$b$$

is bias term.

2.2.2 Regularization in CNN

The target in the learning is to find a concept over the hypothesis space that would cover not only the samples drawn from the training data set, but also the new ones not encountered during the training.

[4, p.225]

For instance instead to recognize the specific image of a cat introduced to the CNN in the training, it would be desired that also other images of a cat could be recognized.

So the ultimate goal is not to learn perfectly the properties of 50.000 images and recognize them by as high accuracy as possible but rather to recognize the similar kind of images outside the training set.

This property is called generalization and much effort has been used for the design of the techiniques to improve it, called regularization.

The main challenge is the model has access only to the empirical distribution of the data, achieved from training data. Since the general rule behind the data would been achieved only if the the data-generating distribution would be known, it should be indirectly deduced by help of training data.

The question how complex the model should be still being able to guarantee the generalization ability, and how much training data is needed for that, is one of the most essential question of the Machine Learning and it is studied in the context of image recognition by CNN in this work experimentally. By increasing the capacity and complexity by increasing the depth and offering new data by applying the DA.

The fundamental problem is to find the balance between the complexity of the learning model and the generalization ability.

The more complexity and capacity it has, more complex rules by combinations of simple functions it can express, but disadvantage is more likely it overlearns non-essential features and less likely finds the general properties of data.

More there are connections between layers the more sensitive the model is also for the noise, and poorer for generalization.

In CNN already the exploiting of prior knowledge known for the structure of data in the design of the architecture and the subsampling of the information by the pooling operations, decreases the risk of the overfitting.

The generalization ability is usually measured by data that does not depend on the training samples, but still is assumed to obey similar probability distribution, that is assumed to be obeyed by the data.

Thus samples can be drawn by identical process, indepedently from the each other, in spite of if they were either in the training or testing set.

In most applications the iid is approximately correct, but in the real-world data it can still be too strong assumption, since the model may need to cover samples from very arbitrary distribution sometimes.

In the CNN of this work two main techniques are applied to increase the generalization ability and to reduce the risk of overfitting.

The effect of first is studied in this work, called Dataset Augmentation, which basicly means the increasing the amount of training data artificially by transforming existing samples to new ones, preserving their class labels unchanged. [5, p.1]

The second one is implemented as a layer of the CNN, called Dropout, which deletes a randomly chosen subset of the connections between layers, thus simplifying the functional depedence between input and output.

The regularization technique chosen to be studied in this work is called Dataset Augmentation. It is very natural to assume that more data is available, easier the model learns useful features that help to solve the classification task.

Since there exists a huge number of different inputs compared to very small limited number of categories, the classifier should be invariant to most of changes for the input.

Thus it is natural to form new samples by transforming the existing ones slightly modified.

For the classification task where for each images there is known the class label it is relatively easy to increase the amount of training data by transforming the existing samples to new ones by some label-preserving transformations.

The basic CNN is not able to achieve the equivariant representation of the images under the scaling, or rotation by the big angles and thus these transformations can be exploited to create artificial training samples from the existing ones.

In Keras there is implemented the tool called ImageDataGenerator, for which can be told which kind of transformations is liked to be used. For example the existing image can be flipped, rotated or shifted, and seen by CNN it is then new training sample.

In Dropout, subset of connections are randomly chosen and their weights set to zero, thus deleting part of the depedencies between input and output. It can be seen as a kind of binary mask for the connections, where 1 means keeping the connection and 0 to delete its effect. [7] For each subset different connections are deleted, thus for each minibatch one of these subsets are chosen to be applied.

This resembles the bagging where different models are trained by splitting the original data set to different subsets.

The ensemble of subnetworks consists of all possible interconnections between output and other units, thus there number is 2^n different possibilities when the n is total number of connections between two layers.

The extreme case is that all connections are set randomly to zero, while there is no more depedence between input and output, which would cause the total forgetting of the data. Other extreme would that every interconnections are non-zero, which would correspond dense connections the MLP.

Because bagging is done randomly, the probability for both cases are extremely low.

In addition if either one of subcases would be picked for single minibatch, its effect is neglible small to the whole training procedure.

3. IMPLEMENTATION AND RESULTS

3.1 Implementation

The experiments of this research were implemented by the high-level Deep Learning software platform Keras.

Its source code is available on Github, and is documented in the home page of Keras. [10]

The CNN for the Cifar-10 classification task has been implemented as an example code, where the number of Convolutional layers has been set to four as default.

Keras needs underhood the backend, which was chosen to be Theano.

Theano optimizes the computational graph defined by Keras, so that its execution would be as efficient as possible. [11]

The basic datastructure in Keras is the Sequential Model, where the layers are organized in the sequential linear order, supporting thus the structure of the feedforward networks.

The CNN of example code consists of four convolutional layers, every second of them is followed by MaxPooling and Dropout layers.

As described in section 2.1, a single convolutional layer consists of three sublayers, a convolution layer, an activation layer and a pooling layer. Since the pooling operation drops the dimension of the feature map, it is added only after every second convolutional layer.

It was chosen to be MaxPooling, pooling area to be 2x2, which means that one fourth of the pixels in image is preserved and other forgotten. [10]

Even if the dimension of the feature map would be non-zero, too much information can be reduced already, and thus it drops the classification.

The Dropout is chosen to drop one fourth of the connections, so only 75 percent of feature values are preserved.

After the convolutional layers, the form of the feature signal is transformed so that it is more suitable for the classification layer. In this point there exist still three color channels, and for each of them own feature vector.

After them the form of the feature map is transformed in Flatten layer, by queuing the separate vectors of three color channels to single one.

Before the classification step there exists Dense layer, which implements densely connected MLP, first returning 512-dimensional vector.

That is mapped through the Activation layer, processed by Dropout that deletes half of connections.

Before experiments it was smartly guessed that the sensible range of the depth is from 4 to 8 layers. If more or less is applied, the dimension of the feature image would drop too much and thus the accuracy.

The final layer before the classification is Dense, which is densely connected MLP, to help to approximate the function by increasing number of parameters.

The final layer is a classifier, the Softmax, which returns from each feature vector the probability vector describing its probability to belong to each of ten categories.

The configuration was conserved otherwise identical to the example code.

Thus the reason for any changes in the performance are most likely due to the modifications of the chosen hyperparameters.

The optimizer defines which of optimizing algorithms is applied in the seeking of minimum for loss.

For Dataset Augmentation there is used data structure called ImageDataGenerator, which takes as parameters the information how the geometrical transformation is done. In this work, images are shifted both in the width and height, 10 percent compared to their sizes, and flipped vertically 180 grades. It can be asked if the CNN really finds them as different images after transforming them in this way or should the transformation be still more complex. [10]

There are also many other options like a horizontal flipping available but the transformations were limited to keep the modifications as slight as possible.

The figure shows example code of Keras for CNN with maximum depth, eight layers, in Cifar-10 image recognition, by which the experiments were started. [14]

3.2 Results

In the hypothesis of this work it is suggested that by changing two hyperparameters of CNN, the learning performance in the image classification can be significantly improved.

The chosen hyperparameters were the depth of the networks, and the use of the Dataset Augmentation.

```
model.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))


model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(128, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))


model.add(Conv2D(256, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(256, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))


model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

***Program 3.1**.]The CNN implemented in Keras, from Keras-examples on Github [14]*

Table 3.1. *The accuracy of CNN without Dataset Augmentation training accuracy*

Depth	**4**	**5**	**6**	**7**	**8**
accuracy	0.8631	0.8627	0.8899	0.8917	0.9179
error	0.3917	0.3961	0.3109	0.3093	0.2316

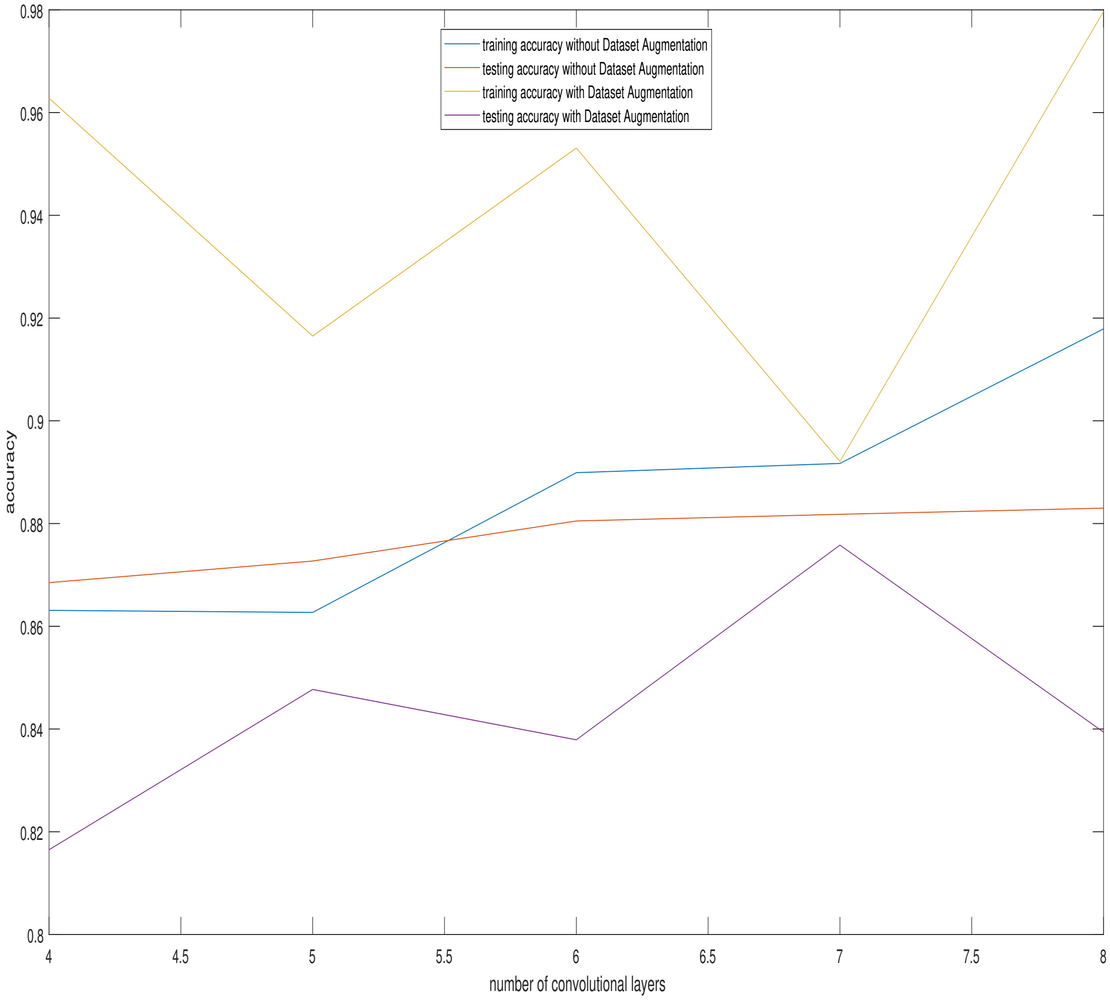
Table 3.2. *The accuracy of CNN using Dataset Augmentation training accuracy*

Depth	**4**	**5**	**6**	**7**	**8**
accuracy	0.9628	0.9165	0.9531	0.8921	0.9797
error	0.1043	0.2330	0.1356	0.3083	0.0602

The accuracy for training is represented in a table for five depths, first for the case where DA was not applied, and then when it was applied. As a consequence the CNN was trained altogether 10 times.

The same results are described in the figure by the graph, the vertical axis is describing the depth, from four to eight convolutional layers and the horizontal axis the accuracy, separately for the training and testing cases. For every four different accuracy there exist graph that has own color, blue describes the training without DA, red the testing without DA ,yellow training with DA and purple testing with DA.

***Figure 3.1**. The training and testing accuracy by depending on the number of convolutional layers and the use of Dataset Augmentation*



4. CONCLUSION

The target of this work was to study if the learning performance of CNN in the image recognition can be improved easily by changing the two hyperparameters. They were chosen to be the depth of CNN and the use of regularization techinique Dataset Augmentation.

The first part of the hypothesis was that the increment in the depth would increase the learning accuracy.

Shown by the figure, and the table it seems keep true for the case where DA was not used.

Both the training and testing accuracy increased when the depth was increased while the DA was not used.

If the DA was applied, the training accuracy increases even more but instead the testing accuracy drops significantly. It is actually at the maximum depth even poorer than by the initial configuration with a minimum depth and without using DA.

In other words the improvement of the accuracy for limited training data is achieved, but not learning for general features, that would improve the solving the learning task. This strongly suggests that CNN overfits the training data if the depth is increased and Dataset Augmentation applied concurrently.

Since the depth means that more abstract features are extracted, and the DA on the otherhand more data is offered it could be suggested that situations where the model has too eager feature extraction in the limits of these experiments it is suggested that there has been extracted too abstract features for too big amount of data, that drops the accuracy.

The applied DA can be too restricted, since only flipping and shift are used. It must be remembered that since CNN is almost shift-invariant, it is possible that only flipping is really working augmentation method from applied ones.

The shifting of images can produce in view point of CNN almost identical data compared to the training data, therefore not really new features have been learned.

The geometrical transformations that are defined by ImageData Generator , could be changed more complex. For instance input and sample means could be set to zero , ZCA whitening could be applied, standard deviation of samples normalized or images rotated. [10]

The interesting question which waits still answer is how for different kind of data the optimal depth of the DNN could be found, amongst other hyperparameters. It seems if otherwise the configuration is kept unchanged, the hidden layers can be added as long as the dimension of the feature map has enough information left.

The use of pooling and dropout drops the dimension at every second layer, thus the eight is the maximum depth for sensible dimension for Cifar-10 image data set.

It is still possible that the dimension of the images have dropped too low, thus too much information is deleted.

It seems better generalization accuracy would be achieved if either depth was increased, or more data was augmented, without applying these changes concurrently.

This rises question, why two modifications that separately applied obviously improve the accuracy does not do it when applied concurrently.

Because CNN was trained only at once by each of configuration of hyperparameters, the random factors may still effect to the results.

To make more reliable conclusions more experiments should be done, maybe testing for different combinations of other hyperparameters too, for example for different optimizing algorithm, or different batch size by which the data is fed.

The reasons can be the SGD converged to the local minimum that was too high for the recognition task, or Dropout hit to the pathological case, where too many weights were set to zero.

Natural development could be to find the mechanism for the algorithm to find optimal hyperparameters values, the depth and regularization, autonomously, even to modify the topology of the networks and other architectural design.

REFERENCES

[1] Duda, R.O., Hart, P.E. , Stork, D.G., Pattern classification, 2001, 2nd ed. Wiley, New York (NY)

[2] Hastie, T., Tibshirani, R. Friedman, J.,The elements of statistical learning: data mining, inference, and prediction, 2009,2nd ed., Springer, New York,

[3] Haykin, S., M. Goossens, J. Braams, D. Carlisle, C. Rowley, Neural networks: a comprehensive foundation, 2nd ed.,Macmillan College Publishing Company, New York

[4] Ian Goodfellow, Yoshua Bengio, Aaron Courville, The Latex Companion, 2nd ed., Deep Learning, MIT Press,2016.

[5] Krizhevsky, A., Sutskever, I. Hinton, G.E., Imagenet classification with deep convolutional neural networks, pp. 1097-1105.,2012.

[6] Mitchell, T.M. , McGraw-Hill, New York,1998

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

[8] Bishop, C.M., Neural networks for pattern recognition,1998, Repr. ed. Clarendon, Oxford,

[9] Bishop, C.M., Pattern recognition and machine learning,2006, 2nd ed. Springer, New York (NY)

[10] F. Chollet, “Keras”,http://keras.io,2018

[11] Theano Development Team , “Theano”,http://deeplearning.net/tutorial/lenet.html,2018

[12] https://deeplearning4j.org/convolutionalnetwork

[13] https://www.cs.toronto.edu/ kriz/cifar.html

[14] F. Chollet, https://github.com/keras-team/keras/blob/master/examples/cifar10_cnn.py/