VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2017 Bc. Ondřej Zapletal

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

ROZPOZNÁVÁNÍ OBRAZŮ KONVOLUČNÍMI NEURONOVÝMI SÍTĚMI - ZÁKLADNÍ KONCEPTY
IMAGE RECOGNITION BY CONVOLUTIONAL NEURAL NETWORKS - BASIC CONCEPTS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. Ondřej Zapletal

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Karel Horák, Ph.D.

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Kybernetika, automatizace a měření**
Ústav automatizace a měřicí techniky

Student: Bc. Ondřej Zapletal ***ID:*** 119681
Ročník: 2 ***Akademický rok:*** 2016/17

NÁZEV TÉMATU:

Rozpoznávání obrazů konvolučními neuronovými sítěmi - základní koncepty

POKYNY PRO VYPRACOVÁNÍ:

1. Seznamte se s pojmem konvoluční neuronové sítě a popište jak teorii tak hlavní aplikační oblasti.
2. Proveďte zevrubnou řešerši možností použití CNN v oblasti rozpoznávání obrazu.
3. Navrhněte strukturu sítě a implementujte algoritmus pro klasifikaci obrazů databáze ImageNet.
4. Proveďte učební a verifikační experimenty pro optimalizaci parametrů sítě z hlediska klasifikační chyby na zadané databázi.
5. Proveďte klasifikaci obrazů z databáze ImageNet a svoje výsledky porovnejte s výsledky posledního ročníku soutěže ILSVCR, porovnejte zejména klasifikační chybu a rychlost učení.

DOPORUČENÁ LITERATURA:

1. GONZALES, R.C., WOODS, R.R.: Digital image processing (3rd edition). Prentice-Hall, Inc., 2008. 954 pages. ISBN 978-0131687288.

2. YOUNG,I.T., GERBRANDS,J.J., VLIET,L.J.: Fundamentals of Image Processing. TU Delft, 1998. 113 pages. ISBN 9075691017.

3. SZELINSKI, R.: Computer Vision: Algorithms and Applications. Springer,2010. ISBN 978-1848829343.

Termín zadání: 6.2.2017 ***Termín odevzdání:*** 15.5.2017

Vedoucí práce: Ing. Karel Horák, Ph.D.
Konzultant:

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

ABSTRACT

This thesis is studying basic concepts of Convolutional Neural Networks. Influence of structural elements on ability of the network to train is investigated. Result of this thesis is comparisons of designed model of Convolutional Neural Network with results from ILSVRC competition.

KEYWORDS

Neural Networks, Convolutional Neural Networks, Machine Learning, Computer Vision, Classification, ImageNet, ILSVRC

ABSTRAKT

Tato práce se zabývá teoretickými principy konvolučních neuronových sítí. V rámci práce je studován vliv struktury konvoluční sítě na její trénování. V závěru práce je schrnuto porovnání dosažených výsledků navrhnutého modeul s výsledky z minulých ročníků soutěže ILSVRC.

KLÍČOVÁ SLOVA

Neuronové Sítě, Konvoluční Neuronové sítě, Strojové Učení, Počítačové Vidění, Klasifickace, ImageNet, ILSVRC

ZAPLETAL, Ondřej. *Image Recognition by Convolutional Neural Networks - Basic Concepts*. Brno, 2017, 68 p. Master's Thesis. Brno University of Technology, Faculty of Electrical Engineering and Communication, Department of Control and Instrumentation. Advised by Ing. Karel Horák, Ph.D.

DECLARATION

I declare that I have written the Master's Thesis titled "Image Recognition by Convolutional Neural Networks - Basic Concepts" independently, under the guidance of the advisor and using exclusively the technical references and other sources of information cited in the thesis and listed in the comprehensive bibliography at the end of the thesis.

As the author I furthermore declare that, with respect to the creation of this Master's Thesis, I have not infringed any copyright or violated anyone's personal and/or ownership rights. In this context, I am fully aware of the consequences of breaking Regulation § 11 of the Copyright Act No. 121/2000 Coll. of the Czech Republic, as amended, and of any breach of rights related to intellectual property or introduced within amendments to relevant Acts such as the Intellectual Property Act or the Criminal Code, Act No. 40/2009 Coll., Section 2, Head VI, Part 4.

Brno ....

author's signature

ACKNOWLEDGEMENT

I would like to thank to my adviser Mr Ing. Karel Horák, Ph.D. for technical leadership, consultations, patience a contributing ideas. I would also like to thank my girlfriend Julia Strašiftáková for her patience and overwhelming support. I couldn't have finish it without you.

Brno ....

author's signature

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Deep Learning and machine learning in general has been hot topics of discussion over last several years. From historical perspective this is very interesting because research in artificial intelligence started in mid-20th century with bold promises that did not come into fruition when the century ended. It was not the case that the research failed to bring any fascinating innovations but it merely failed to bring machines that could see or to understand human voice. There were many reasons for this but perhaps the most important thing is that both vision and voice recognition are much more complex that our own intuition would conceive since it comes to us very naturally.

The long-awaited breakthrough came about ten years later in application of neural networks. It came somewhat unexpected at times when large portion of research community stopped paying attention to them. Neural networks are one of the oldest models that many people perceive as failed endeavor. From year 2010, sources of technical news are constantly coming with new and exciting stories of deep learning application in almost all walks of life. Leading Fortune 500 companies such as Amazon, Apple, Facebook or Google are investing enormous resources into machine learning research.

Technology that we use today is taking full advantage of this recent development. Mobile devices in our pockets are capable of identifying specific human face. They are capable to understand and react to voice commands. Self-driving cars are most likely about to become part of our every day life.

In the midst of all this is one specific model that is most likely the largest contributor to current state of affairs. This model is called Convolutional Neural Network (CNN). Aim of the thesis is to document its background and basic theoretical concepts. Several examples of applications of CNN will be discussed. Structure of CNN model will be proposed in one of the available software tools. Resulting model will be tested on Large Scale Visual Recognition Challenge (ILSVRC) dataset and its results will be compared with submissions from previous years' participants.

2 THEORY

This chapter is focused on defining concepts of artificial intelligence, image processing and machine learning in general.

2.1 Artificial Intelligence

Field of Artificial Intelligence (AI) is very general and spawns across several different disciplines (e.g. mathematics, computer science, philosophy, economics and even ethics). This fact suggests that this field is very broad and can be tackled from many different view points. Therefore, this description will not be very exhaustive. For broader and more complex dive into subject refer to [28].

One of many possible definitions of AI can be summarized as a pursuit to develop an artificial intelligent agent. In other words, it is an attempt to create intelligent machines that are either thinking or can be perceived as thinking ones. One of the most important abilities of intelligent agents is the sense of vision[1]. Sense of vision is usually required to certain degree and not always it is necessary that it rivals the capabilities of human visual apparatus.

First attempts to solve the vision problems were tackled from so called bottom-up approach in which the system was instructed with hard-coded set of rules describing the vision. It was expected that as the understanding of a mechanisms allowing humans to extract information from visual scene, the hard-coded systems can be fed this understanding and thus more capable systems can be created. Problem with this approach was that it highly underestimated the difficulty of formalization of these rules. As a consequence it predominantly failed to devise such a system.

This insight leads the researchers to postulating that in order to solve the problem of deploying vision capabilities for artificial intelligent system, it is necessary to introduce a process that would allow AI systems to extract patterns from provided data. That is to say, it is an introduction of systems that are able to learn. Process that enables systems to learn is generally called machine learning.

Machine learning is again quite extensive term that can be used in multiple different contexts. In this work, it is meant to be understood as technique that is used to create mathematical models used for image recognition. There are several types of machine learning models that are useful for different tasks. The task that is discussed in this work and is also arguably most commonly tackled is called classification, which is the task to classify instance of input into correct discreet and mainly finite class. Another common type of machine learning task is called

[1] This is highly dependent on concrete application.

regression, which is based on the input data trying to estimate unknown continues-valued quantity.

Rest of this document will be exclusively dealing with learning tasks of classification type.

2.2 Image Processing

Computer vision is important subject in image processing. Research in the subject was approaching the problem from bottom-up perspective for many years. This was the before-mentioned effort to formalize rules guiding the vision of living organisms. This approach was actually very successful in certain circumstances.

General description of computer vision in image processing can be summarized in following steps:

- Image capture - Image is captured (by camera or similar device) and digitized.
- Pre-processing - Digitized image is modified to emphasize important features (e.g. noise reduction, contrast normalization etc.).
- Segmentation - Selection of interesting features (edges, similar surfaces).
- Description - Extraction of radiometric, photometric descriptors and so on.
- Classification - Some means to classify the object.

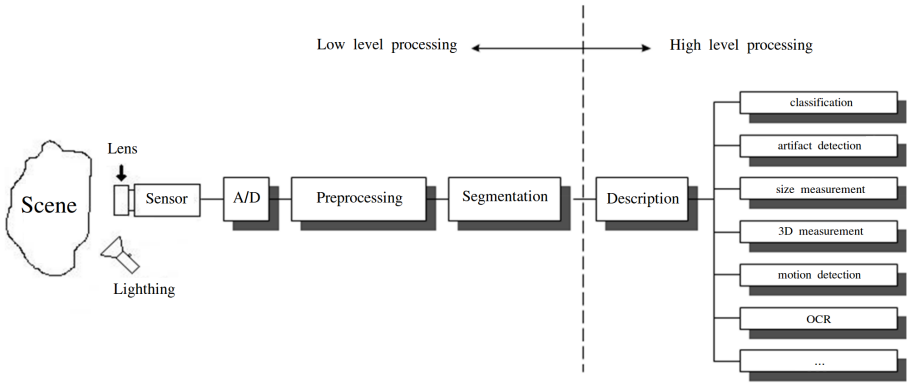

Figure 2.1: Block diagram of image processing pipeline. [20]

Individual steps are shown in figure 2.1. Even though machine learning existed as field of study since the second half of the 20th century, there was no wider adoption of its techniques in image processing for very long time. It was first introduced in the classification step of the processing pipeline. In other words, complex problems were simplified by reducing the visual information contained within the image

into handful of simple features that were fed into machine learning model. This approach consequently favors very simple models such as K-nearest Neighbors (KNN) or Support Vector Machines (SVM).

This brings the problem that these applications are not very versatile. Each application is usually only capable of solving very narrow problem and any deviation from ideal circumstances can mean failure. Application can have problems with varying contrast, illumination, scaling, rotation etc.

Second problem is often the fact that because image has to be pre-processed several times before it is fed into machine learning model it requires extra time and computation resources. This is less of a problem with current hardware innovations, but it is still not negligible factor and it can have negative effect on the cost of the solution. This is where machine learning in general indicates significant advantage.

Classical approach to computer vision can find success in applications that are deployed in very restricted environments with rigid constraints[2]. In controlled environment, it is usually very simple to describe the problem in formal rules. Even though it can be still found in certain places, it starts to be forced out by application of machine learning simply because the barrier of entry for wider adoption decreases every day.

2.3 Machine Learning

As previously described, Machine Learning is a process that is used to create models that are able to extract information from data to solve given problem and consequently automatically improves their performance.

Interesting perspective that can be used is to view machine learning as form of information compression. Where machine learning model is trying to extract information from input data in such a way that the amount of a data used to save is reduced while the information contained within is preserved.

2.3.1 Machine Learning Approach

There are typically two different types of machine learning approaches:

- Unsupervised Learning
- Supervised Learning

Both are typically used for different kinds of machine learning tasks.

[2]This could be for example detection of defects on line production in industrial automation.

Unsupervised learning

In this learning approach, the model is training by observing new data and extracting patterns in the date without being instructed on what they are. Opposed to supervised learning described below, the advantage of this approach is that the model is able to learn from data without supervision (as the name suggests). This means that there is no need for input data to be annotated, therefore it takes much less time and resources to deploy these models in practice.

The biggest hurdle of supervised learning approach in real world applications is to obtain appropriate data. Appropriate data in this context means, data that were somehow classified into different categories, which can be very tedious and slow process. In some cases, the task itself prevents the usage of labeled data (i.e. labeled data are impossible to obtain or don't exist at all).

Majority of unsupervised learning algorithms belong to group called clustering algorithms. These algorithms are centered on the idea to analyze geometric clustering of data in input space to determine their affiliation. This is achieved by the presupposition that data point clustering in input space are likely to exhibit similar properties.

Examples of unsupervised learning models are:

- K-means - clustering model [17, p. 460–462];
- Self Organizing Maps (SOMs) - instance based [22];
- Principal Component Analysis (PCA) - dimensionality reduction [17, p. 534–544].

Image classification usually does not rely heavily on the use of unsupervised learning methods, therefore the following text describes only supervised learning methods.

Supervised learning

Supervised learning approach is more commonly used. This approach requires training data with specific format. Each instance has to have assigned label. These labels provide supervision for the learning algorithm. Training process of supervised learning is based on the following principle. Firstly, the training data are fed into the model to produce prediction of output. This prediction is compared to the assigned label of the training data in order to estimate model error. Based on this error the learning algorithm adjusts model's parameters in order to reduce it.

2.3.2 Structure of Machine Learning Algorithm

Although machine learning algorithms are diverse and are using different techniques its structure can be generalized. Structure of nearly all machine learning algorithms can be described as composition of the following components:

- Dataset specification
- Model
- Cost function
- Optimization procedure

Almost all supervised learning algorithms use the same Dataset specification. The other three components can vary dramatically. This level of analysis is useful for building of intuition for Neural Networks (NNs) and explanation of its individual components.

Dataset specification

Supervised learning requires datasets with specific properties. Each dataset contains set of n instances which consists of a pair of input vector $\boldsymbol{x}_i$ and output scalar y_i. Input vector

$$\boldsymbol{x}_i^T = [x_1, x_2, \ldots, x_p], \tag{2.1}$$

where i is index of instance, p, is dimension of input vector.

Individual components of input vector have to be of unified type. In case of input data in form of image it is value for individual pixels (e.g. 0-255). In other cases, they can be real values. Almost universally in machine learning it stands that input should be normalized. This presumption holds in images automatically since each pixel has to have its vales in fixed range. It is very important in other types of machine learning tasks, where this is not guaranteed.

Output scalar y_i represents class of given instance. Type of this output value thus has to acquire only certain values. To put it differently, it has to be a set of cardinality equal to number of all possible classes.

Model

Model is prediction apparatus that takes input $\boldsymbol{x}_i$ to predict value of it's output y_i. Each model has parameters represented by vector $\boldsymbol{\theta}$, which are adjusted during the training process. The simplest example of model type is linear model, also called linear regression.

Parameters $\boldsymbol{\theta}$ of this model are

$$\boldsymbol{\theta}^T = [\theta_1, \theta_2, \ldots, \theta_p], \tag{2.2}$$

where p is number of parameters equal to size of input vector $\boldsymbol{x}_i$.

Prediction $\hat{y}_i$ of the model on instance i is computed as

$$\hat{y}_i = \sum_{j=1}^{p} x_{ij}\theta_j. \tag{2.3}$$

Therefore predictions of the model on the entire dataset in matrix notation is

$$\hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\theta}. \tag{2.4}$$

Predictions in expanded notation are equal to

$$\begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_n \end{bmatrix} = \begin{bmatrix} x_{11} & \cdots & x_{1p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{np} \end{bmatrix} \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_p \end{bmatrix}. \tag{2.5}$$

Cost Function

To achieve the learning ability of the machine learning algorithm, it is necessary to estimate the error of its predictions. This is estimated with so called cost function (also sometimes called loss function).

This function has to have certain properties. Ability of the machine learning algorithm to learn rests on the estimation of its improvement with change of its parameters. Therefore, cost function has to be at least partially differentiable. In case of linear regression it is most common to use sum of square error. The main reason being that derivative of this function for linear model has only one global minimum.

Cost function is defined as

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=0}^{n} \left(y_i - \boldsymbol{x_i}^T\boldsymbol{\theta}\right)^2. \tag{2.6}$$

For the optimization purposes it is usually useful to express the cost function in matrix notation

$$J(\boldsymbol{\theta}) = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})^T (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}). \tag{2.7}$$

Optimization Procedure

The last part of learning algorithm is the optimization procedure. It consist of update of model's parameters $\boldsymbol{\theta}$ in order to improve it's prediction. In other words to find $\boldsymbol{\theta}$ such that the value of cost function $J(\boldsymbol{\theta})$ for given dataset is as small as possible.

To investigate the change of cost function on given dataset it is necessary to compute the derivative of $J(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$

$$\begin{aligned}\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} &= \frac{\partial}{\partial \boldsymbol{\theta}} \left[(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})^T (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}) \right] \\ &= \frac{\partial}{\partial \boldsymbol{\theta}} \left[\boldsymbol{y}^T \boldsymbol{y} + \boldsymbol{\theta}^T \boldsymbol{X}^T \boldsymbol{X}\boldsymbol{\theta} - 2\boldsymbol{y}^T \boldsymbol{X}\boldsymbol{\theta} \right] \\ &= 2\boldsymbol{X}^T \boldsymbol{X}\boldsymbol{\theta} - 2\boldsymbol{X}^T \boldsymbol{y}.\end{aligned} \tag{2.8}$$

For linear model is possible to find optimal solution which is global minimum of the cost function. The optimal solution

$$\boldsymbol{\theta} = \left(\boldsymbol{X}^T \boldsymbol{X} \right)^{-1} \boldsymbol{X}^T \boldsymbol{y}, \tag{2.9}$$

is found by equating the partial derivative of $J(\boldsymbol{\theta})$ to 0. Only condition is that $\boldsymbol{X}^T \boldsymbol{X}$ has to be non singular.

Unfortunately, only very simple problems can be approximated using model as simple as linear regression. More complex model usually means more complicated cost function. Optimization process of more complex cost functions cannot be guaranteed to find global minimum. In this case, the optimization procedure has to be of iterative character. To put it in a different way, algorithm has to approach the minimum of iterations. Many of the iterative methods belong to the group called gradient based optimization.

2.3.3 Model Complexity

In the first approximation it could be said that the task of supervised machine learning is to model relationship between the input output data most accurately. The problem with this definition is that in the practical application there is never enough data to capture true relationship between the two. Therefore, the task of machine learning is the attempt to infer true relationship by observing incomplete picture.

Hence the most important property of machine learning model is its generalization ability. That is ability to produce meaningful results from data that were not previously observed.

Generalization ability is dependent on complexity of the model and its relationship to complexity of underling problem. When model does not capture complexity of the problem sufficiently it is described as under fitting. In case the complexity of model exceeds the complexity of underling problem then this phenomenon is called over fitting.

In both of these extremes the generalization ability suffers. In the former case the model is unable to capture true intricacies of the problem and therefore is unable

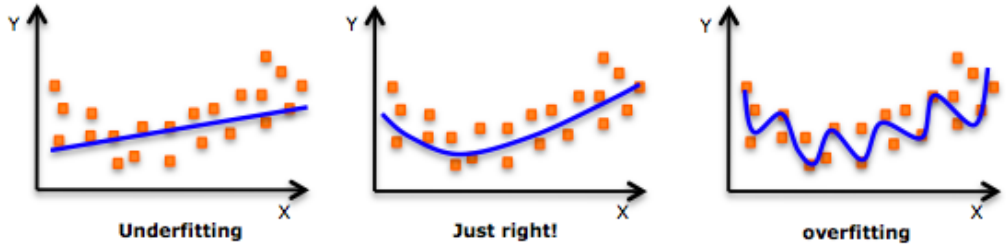

Figure 2.2: Figure shows different levels of generalization of model [1]

to predict desired output reliably. In the latter case it tries to capture even the subtlest data perturbation that might be in fact a result of stochastic nature of the problem and not the real underlying relationship. This can also cause the fact that input data is missing some variable necessary to capture the true relationship. This fact is unavoidable and it thus has to be taken into account when designing machine learning model. Depiction of this phenomena in case of two variable inputs is on Figure 2.2.

Typically, the machine learning model is trained on as much input data as possible in order to achieve the best possible performance. At the same time its error rate has to be verified on independent input data to check whether the generalization ability is not deteriorating. This is typically achieved by splitting available input data into training and testing set (usually in 4:1 ratio for training to test data). Model is trained with training data only and the performance of the model is tested on the test data. Relationship between test and train error can be found on Fig. 2.3. Even though the true generalization error can never be truly observed, its approximation by test error rate is sufficient for majority of machine learning tasks.

Regularization

Regularization is any modification that is made to the learning algorithm that is intended to reduce its generalization error but not its training error [16].

As it has already been mentioned, the most important aspect of machine learning is striking the balance between over and under fitting of the model. To help with this problem concept of regularization was devised. It is a technique that helps to penalize the model for its complexity. Basic concept consists of adding a term in the cost function that increases with model complexity.

When this is applied to cost function from equation 2.7

$$J(\boldsymbol{\theta}) = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})^T (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}) + \lambda \boldsymbol{\theta}^T \boldsymbol{\theta}, \tag{2.10}$$

where λ is a parameter that controls the strength of the preference [16].

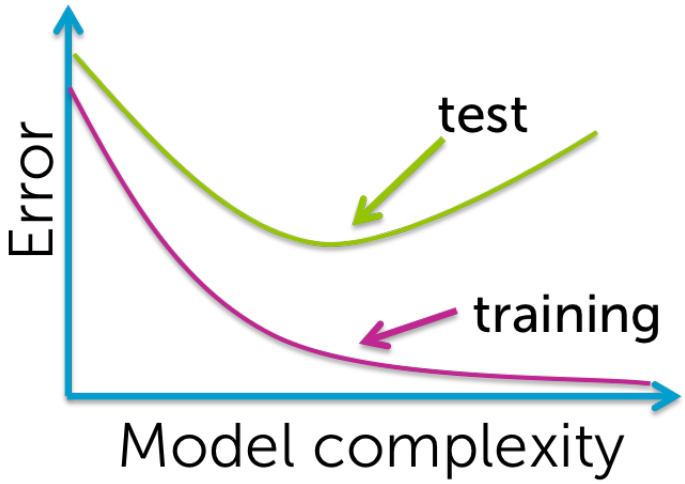

Figure 2.3: Relationship between the model complexity and its ultimate accuracy is the relationship between training and testing error [7].

3 NEURAL NETWORKS

This chapter is dedicated to description of NN in general and its special type called CNN.

3.1 History

History of NNs can be arguably dated from 1943, when Warren Mcculloch and Walter Pitts devised mathematical model inspired by Biology of central nervous systems of mammals [25].

This inspired the invention of Perceptron, created in 1958 by Frank Rosenblatt. Perceptron used very simple model mimicking biological neuron that was based on mathematical model of Pitts and Mcculloch. Definition of the Perceptron model also described an algorithm for direct learning from data.

In the beginning Perceptron seemed very promising, but it was soon discovered that it had severe limitations. Most prominent voice of criticism was Marvin Minsky. Minsky published book in which he laid out a case that Perceptron model was unable to solve complex problems [26]. Among others the book contained mathematical proof that Perceptron is unable to solve simple XOR problem. More generally the Perceptron is only capable of solving linearly separable problems. Even though according to Minsky this criticism wasn't malicious, it in effect stifled the interest in NNs for over a decade.

Interest in NNs was rejuvenated in the early 80's, when it was shown that any previously raised deficiencies could have been solved by usage of multiple units. This was later exacerbated by invention of back-propagation learning algorithm, which enabled the possibility to gather neurons into groups called layers, which can be stacked into hierarchical structures to form a network. NN of this type were commonly called Multilayer Perceptron (MLP).

In 80s and 90s the interest in NNs plateaued again and general research of AI was more focused on other[1] machine learning techniques. In the realm of classification problems, it were notably SVM and ensemble model. AI research community also developed several other paradigms of NNs that were similarly inspired by Biology of certain aspect of central nervous system but took different approaches. Most important examples were SOM and Recurrent Neural Network (RNN) (e.g. hopfield networks).

By the year 2000, there was very few research groups that were devoting enough attention to the NNs. There was also certain disdain for NNs in academia and AI research community. Success of NNs that was promised almost half a century ago

[1]These models were usually less complex than NNs

was finally encountered around 2009, when the first networks with large number of hidden layers were successfully trained. This led to mainstream adaption of umbrella term deep learning which by and large refers to Deep Neural Network (DNN). The word deep indicates that networks have large number of hidden layers.

The key theoretical insight was to learn complicated functions that could represent high-level abstractions (e.g. vision recognition, language understanding etc.). There is a need for deep architecture.

NNs in the times before DNNs had only 1 or 2 hidden layers. These are today often called shallow networks. Typical Deep Networks can have number of hidden layers in order of tens, but in some cases even hundreds [18]

Even though that progress of Neural Network into direction of structures with high number of hidden layers was obvious, its training was unsolved technical problem for very long time. There were basically 3 reasons why this breakthrough didn't come sooner.

1. There were no technique allowing the number of hidden layers to scale.
2. There wasn't enough of labeled data necessary to train the NN.
3. The computational hardware wasn't powerful enough to train sufficiently large and complex networks effectively.

First problem was tackled by invention of CNNs [24]. Second problem was solved simply when there was more data available. This was mainly achieved thanks to effort of large companies (Google, Facebook, YouTube, etc.) but also with help of large community of professionals and hobbyists in data sciences.

Both innovation in computational hardware and improvement of training methods were needed to solve the third problem. One of the technical breakthroughs was utilization of Graphics Processing Units (GPUs) for the demanding computation involved in training of a complex network. Thanks to the fact that training process of NNs is typically large number of simple consequent computations, there is a great potential for parallelization.

3.2 Structure of Neural Networks

The term NN is very general and it describes broad family of models. In this context NN is distributed and parallel model that is capable of approximating complex non-linear functions. Network is composed from multiple computational units called neurons assembled in particular topology.

Description of NN structure will follow the convention laid out in the description of learning algorithm. Meaning that a description of the learning algorithm is composed of model, cost function and optimization procedure. The difference comes into play with the fact that model of NN is much more complex than the

model linear regression. Therefore, the analysis is divided into model of neuron and topology of the network.

3.2.1 Model of Neuron

Neuron is computational unit performing nonlinear transformation of its inputs

$$y = g(\boldsymbol{w}^T \boldsymbol{x} + b). \tag{3.1}$$

Argument $\boldsymbol{w}^T \boldsymbol{x} + b$ of function g is often regarded as z. Therefore, the equation can be rewritten as

$$y = g(z). \tag{3.2}$$

Typical schema is shown on Figure 3.1, which depicts inputs, weights bias and activation function.

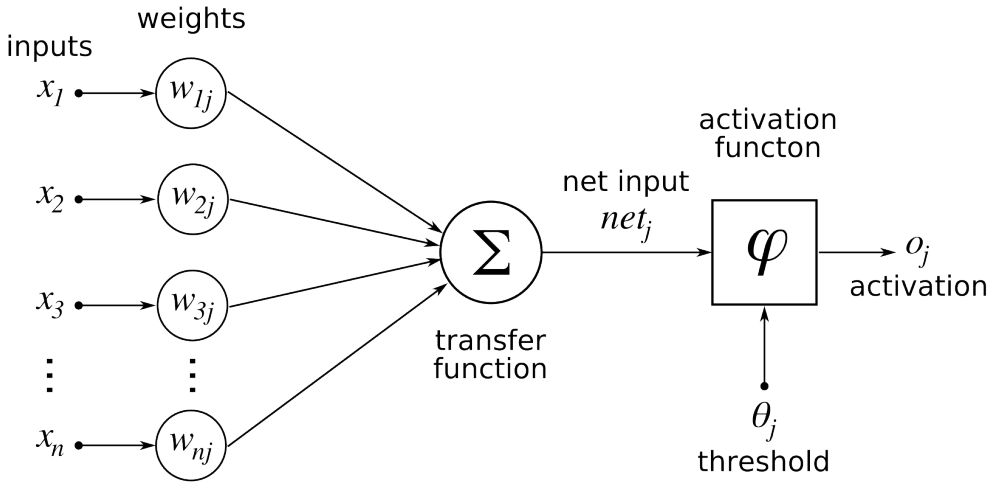

Figure 3.1: Model of artificial neuron [29].

As it was already mentioned model of neuron was inspired by biology. First attempts to create model of neuron had multiple elements equivalent with neurons of human brain. As research progressed this equivalence ceased being as important and modern NN models correspond to their biological counterparts only superficially.

Inputs

Each neuron has multiple inputs $\boldsymbol{x}$ that are combined together to execute some operation. Each input has designated weight assigned to it.

Weights

Inputs of a neuron are weighted by parameters $\boldsymbol{w}$ that are modified during learning process. Each weight gives strength to each individual input into the neuron. The

basic idea is that when the weight is small the particular input doesn't influence the output of the neuron very much. Its influence is large in the opposite case.

Bias

Another modifiable parameter is bias b that controls influence of the neuron as a whole.

Activation Function

For NN to approximate nonlinear function each neuron has to perform nonlinear transformation of its input. This is done with activation function $g(z)$ that preforms nonlinear transformation. There are several different commonly used activation functions. Its usage depends on the type of network and also on the type of layer in which they operate.

One of the oldest and historically most commonly used activation function if sigmoid function. It is defined by

$$g(z) = \frac{1}{1 + e^{-z}}. \tag{3.3}$$

Problem with sigmoid is that its gradient becomes really flat on both extremes and as such it slows down the learning process [23].

Another activation function is hyperbolic tangent. It is defined as

$$g(z) = tanh(-z). \tag{3.4}$$

Hyperbolic tangent function is less common in feed forward NN, but it is largely used in RNN.

Currently most frequently used activation function is Restricted Linear Unit (ReLU). It is very commonly used in both convolutional and fully connected layers. It is defined by

$$g(z) = \max\{0, z\}. \tag{3.5}$$

It has a drawback because it is not differentiable for $z = 0$, but it is not a problem in software implementation and one of its biggest advantages is that it can learn very quickly.

All three activation functions are illustrated in Figure 3.2.

3.2.2 Topology of the Network

There are several different commonly used topologies. Two most commonly used in deep learning are feed-forward and recurrent. Feed forward networks are characterized by the fact that during activation the information flows only in forward

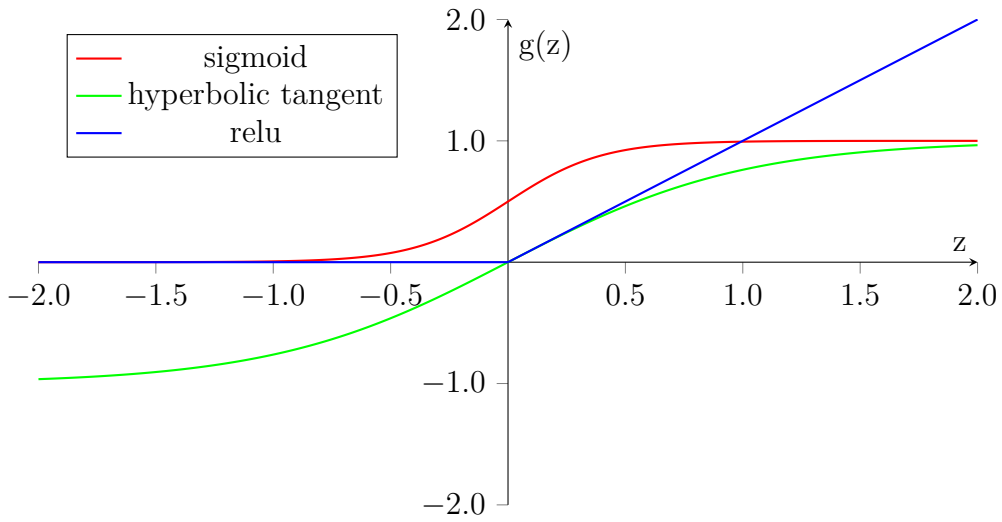

Figure 3.2: Activation Functions

direction from inputs to output. A recurrent network has some sort of feedback loop.

Another criterion of topology is how are individual neurons in the network connected. Most commonly are NNs ordered in layers. In each layer there can be from *1* to n neurons. Layers are hierarchically stacked. In typical terminology the first layer is called input layer, the last layer is called output layer and the layers in-between are called hidden.

Description of the network rests on interconnections between individual layers. Most common scheme is called fully connected where each neuron in hidden layer l has input connections from all neurons from previous layer $l-1$ and its output is connected to input of each neuron in following $l+1$ layer. Entire structure is illustrated on Figure 3.3.

From this point on the term NN will refer to Feed-forward Fully Connected Neural Network.

Types of neurons are dependent on the type of the layer. Currently the main difference is in their activation function, which wasn't the case for a long time. Historically all layers had neurons with sigmoid activation function. It was mainly because the output sigmoid layer can be easily mapped onto probability distribution, since it acquires vales between 0 and 1. Only relatively recently[2] it was found that network composed of neurons with ReLU activation function in the hidden layers can be trained very quickly and are more resistant against over-fitting. Activation functions are still subject of ongoing research.

Neurons in output layer need output that can produce probability distribution

[2]In the last decade which is relatively recently in the grand scheme of NN history.

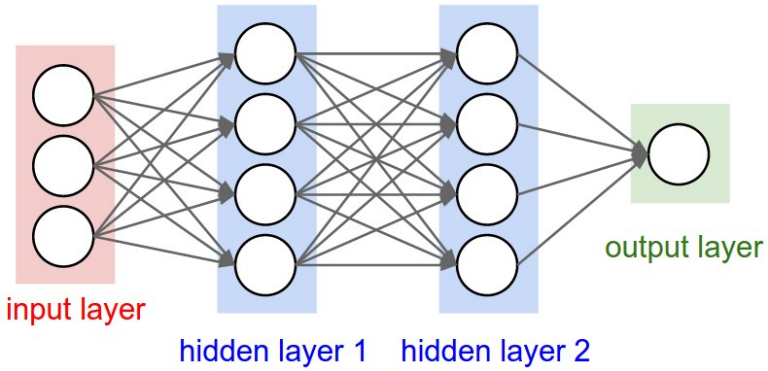

Figure 3.3: Fully connected Feed Forward Neural Network [21].

that can be used to estimate the probability of individual classes. For this reason, most commonly used activation function of output neuron is called softmax. Softmax is normalized exponential function. It is used to represent probability of an instance being member of class j as

$$g(z)_j = \frac{e^{z_j}}{\sum_K^{k=1} e^{z_k}}, \tag{3.6}$$

where K is total number of classes.

3.2.3 Cost Function

Cost functions of NNs is a complex topic that exceeds scope of this thesis. One of the most common cost function used in NNs for classification into multiple classes is categorical cross entropy. For softmax activation function from Equation 3.6 is cost function defined as

$$C = -\frac{1}{n}\sum_{i=1}^{n} y^{(i)} \ln g(z^{(i)}) + (1 - y^{(i)}) \ln(1 - g(z^{(i)})), \tag{3.7}$$

where $y^{(i)}$ if correct class of the instance and n is total number of instances.

3.2.4 Optimization Procedure

Every optimization procedure for NN is based on gradient descent. In other words, it is iterative process that aims to lower training error of the network by differentiating of cost function and adjusting parameters $\boldsymbol{\theta}$ of the model by following the negative gradient.

The problem is that cost function of entire network is very complex and has many parameters. To find the gradient of the cost function it is necessary to go through all of the units in the network and estimate their contribution to the overall error. Technique that is used to solve this problem is called back-propagation.

Back-propagation if often confused to be complete learning algorithm which is not the case, it is only the method to compute the gradient [16].

Back-propagation

To estimate the influence of individual units in a network the back-propagation is used to compute delta δ_j^l, where l is layer and j is index of neuron in that layer. Algorithm starts at the output of NN, more specifically its cost function.

$$\delta^L = \nabla_x C \odot g^{'}(z^L) \tag{3.8}$$

where L is last layer of the network and $\nabla_x C$ is gradient of cost function with respect to x and $\odot$ is the Hadamard product[3].

In subsequent lower layers the deltas are computed as

$$\delta^l = ((\boldsymbol{w}^{l+1})^T \delta^{l+1} \odot g^{'}(z^l) \tag{3.9}$$

where $(\boldsymbol{w}^{l+1})^T$ is from Equation 3.1.

Each neuron has two modifiable parameters b and $\boldsymbol{w}$. To estimate the rate of change for parameter b_j^l from Equation 3.1 it needs to be computed as

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{3.10}$$

Change of weight w_{jk}^l from Equation 3.1 it needs to be computed as

$$\frac{\partial C}{\partial w_{jk}^l} = x^{l-1} \delta_j^l \tag{3.11}$$

Gradient Descent Optimization

Back-propagation estimates gradient of all modifiable parameters b and $\boldsymbol{w}$ in the network. These parameters can be referred to by vector $\boldsymbol{\theta}$. Therefore, the gradient of the function to be minimized can be written as $\nabla_{\boldsymbol{\theta}_{t-1}} f(\boldsymbol{\theta_{t-1}})$.

Simplest learning algorithm is called gradient descent. Even though simple, it is very robust learning algorithm.

$$\boldsymbol{g}_t \leftarrow \nabla_{\boldsymbol{\theta}_{t-1}} f(\boldsymbol{\theta}_{t-1}) \tag{3.12}$$

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta \boldsymbol{g}_t \tag{3.13}$$

Algorithm has one meta-parameter η, which is often called learning rate. It determines how quickly are $\boldsymbol{\theta}$ parameters updated. Simple gradient descent has the shortcoming that update of parameters is always exactly proportional to change

[3]It is element-wise product of matrices.

of gradient. This might become a problem when the gradient change slows down. This algorithm is also often called Stochastic Gradient Descent (SGD). The word stochastic indicates that during training the algorithm is using random selection of instances to train.

There are many different variations on the gradient descent method. Following definitions are taken from [14].

Adam It is more complex learning algorithm that combines L_2 norm and classical momentum based optimization. It should converge faster than classical Gradient Descent.

$$\boldsymbol{g}_t \leftarrow \nabla_{\boldsymbol{\theta}_{t-1}} f(\boldsymbol{\theta}_{t-1}) \tag{3.14}$$

$$\boldsymbol{m}_t \leftarrow \mu \boldsymbol{m}_{t-1} + (1-\mu)\boldsymbol{g}_t \tag{3.15}$$

$$\hat{\boldsymbol{m}}_t \leftarrow \frac{\boldsymbol{m}_t}{1-\mu^t} \tag{3.16}$$

$$\boldsymbol{n}_t \leftarrow \nu \boldsymbol{n}_{t-1} + (1-\nu)\boldsymbol{g}_t^2 \tag{3.17}$$

$$\hat{\boldsymbol{n}}_t \leftarrow \frac{\boldsymbol{n}_t}{1-\nu^t} \tag{3.18}$$

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta \frac{\hat{\boldsymbol{m}}_t}{\sqrt{\boldsymbol{n}_t} + \varepsilon} \tag{3.19}$$

Nadam Nadam is further improvement of Adam that extends it with Nesterov acceleration trick that should in most cases improve speed of convergence [14].

$$\boldsymbol{g}_t \leftarrow \nabla_{\boldsymbol{\theta}_{t-1}} f(\boldsymbol{\theta}_{t-1}) \tag{3.20}$$

$$\hat{\boldsymbol{g}}_t \leftarrow \frac{\boldsymbol{g}_t}{1-\prod_{i=1}^{t}\mu_i} \tag{3.21}$$

$$\boldsymbol{m}_t \leftarrow \mu \boldsymbol{m}_{t-1} + (1-\mu)\boldsymbol{g}_t \tag{3.22}$$

$$\hat{\boldsymbol{m}}_t \leftarrow \frac{\boldsymbol{m}_t}{1-\prod_{i=1}^{t}\mu_i} \tag{3.23}$$

$$\boldsymbol{n}_t \leftarrow \nu \boldsymbol{n}_{t-1} + (1-\nu)\boldsymbol{g}_t^2 \tag{3.24}$$

$$\hat{\boldsymbol{n}}_t \leftarrow \frac{\boldsymbol{n}_t}{1-\nu^t} \tag{3.25}$$

$$\bar{\boldsymbol{m}}_t \leftarrow (1-\mu_t)\hat{\boldsymbol{g}}_t + \mu_{t+1}\hat{\boldsymbol{m}}_t \tag{3.26}$$

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta \frac{\bar{\boldsymbol{m}}_t}{\sqrt{\boldsymbol{n}_t} + \varepsilon} \tag{3.27}$$

3.3 Convolutional Neural Networks

CNNs are specialized type of NNs that was originally used in image processing applications. They are arguably most successful models in AI inspired in biology.

Even though they were guided by many different fields, the core design principles were drawn from neuroscience. Since their success in image processing, they were also very successfully deployed in natural language and video processing applications.

Aforementioned inspiration in biology was based on scientific work of David Hubel and Torsten Wiesel. Neurophysiologists Hubel and Wisel, investigated vision system of mammals from late 1950 for several years. In the experiment, that might be considered little gruesome for today's standards, they connected electrodes into brain of anesthetized cat and measured brain response to visual stimuli [19]. They discovered that reaction of neurons in visual cortex was triggered by very narrow line of light shined under specific angle on projection screen for cat to see. They determined that individual neurons from visual cortex are reacting only to very specific patterns in input image. Hubel and Wiesel were awarded the Nobel Prize in Physiology and Medicine in 1981 for their discovery.

In the following text is presumed that convolutional layer is working with rectangular input data (e.g. images). Even though the Convolutional networks can also be also used to classify 1-dimensional[4] or 3-dimensional [5] input.

3.3.1 Structure of CNN

Structure of Convolutional networks is typically composed of three different types of layers. Layer can be either Convolutional, Pooling or fully connected. Each type of layer has different rules for forward and error backward signal propagation.

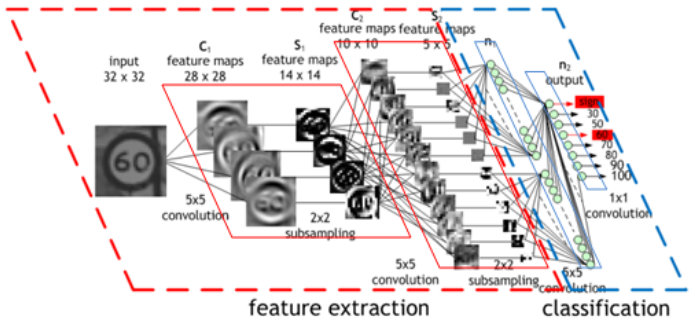

Figure 3.4: Typical structure of Convolutional Neural Network [5]

There are no precise rules on how the structure of individual layers should be organized. However with exception of recent development[6] CNNs are typically structured in two parts. First part, usually called feature extraction, is using combinations of convolutional and pooling layers. Second part called classification is using

[4]For example sound signal.
[5]For example CT scans.
[6]GoogLeNet described in section 4.2.

fully connected layers. This is illustrated in Figure 3.4.

Convolutional layer

As the name suggests this layer employs convolution operation. Input into this layer is simply called input. Convolution operation is performed on input with specific filter, which is called kernel. Output of convolution operation is typically called feature map.

Input into Convolutional layer is either image (in case of first network layer) or feature map from previous layer. Kernel is typically of square shape and its width can range from 3 to N pixels. Feature map is created by convolution of kernel over each specified element of input. Convolution is described in more detail in section describing training of CNN.

Depending on the size of kernel and layer's padding preferences the process of convolution can produce feature map of different size than input. When the size of output should be preserved it is necessary to employ zero padding on the edges of input. Zero padding in this case has to add necessary amount of zero elements around the edges of input. This amount is determined by

$$p = ((h - 1)/2) \tag{3.28}$$

where h is width of used kernel. In opposite case the feature map is reduced by the $2p$. Reduction of size of the feature map can be in some cases desirable. Zero padding is illustrated on Figure 3.5.

Reduction of feature map can go even further in case of use of stride. Application of stride specifies by how many input points is traversed when moving to neighboring position in each step. When the stride is 1, kernel is moved by 1 on each step and the resulting size of feature map is not affected.

Each Convolutional layer is typically composition of several different kernels. In other words, output of this layer is tensor containing feature map for each used kernel. Each of these is designed to underline different features of input image. In the first layers these features are typically edges. Higher the layer, the more complex features are captured.

Each kernel that is used is applied to all inputs of the image to produce one feature map which basically means that neighboring layers are sharing the same weights. This might not be sufficient in some applications and therefore it is possible to use two other types of connections. Locally connected which basically means that applied kernel is of the same size as the input and tiled convolution which means alternation of more than one set of weights on entire input.

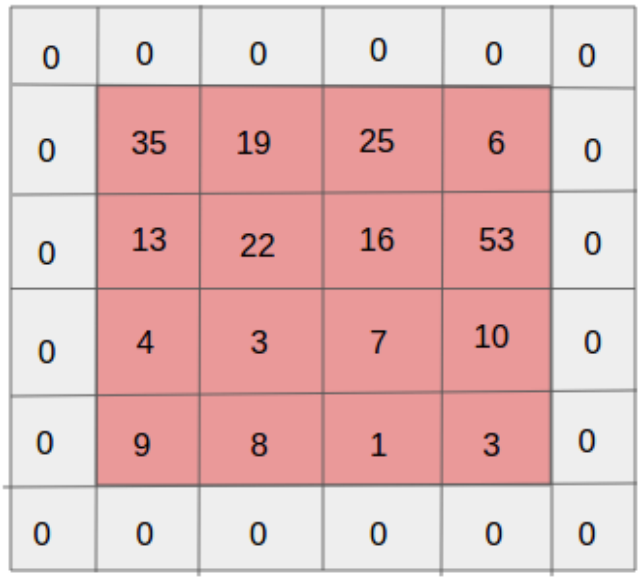

Figure 3.5: A zero padded 4x4 matrix [11]

Tiled convolution is interesting because with clever combination with Max-Pooling explained bellow it allows to train specific feature from multiple angles (in other words invariant to rotation).

Each convolutional layer has non-linearity on its output that is sometimes also called the detector stage. This is equivalent to activation function of NNs. Activation function of CNN is commonly ReLU.

Pooling layer

This layer typically doesn't constitute any learning process but it is used to down-sample size of the input. The Principle is that input is divided into multiple not-overlapping rectangular elements and units within each element are used to create single unit of output. This decreases the size of output layer while preserving the most important information contained in input layer. In other words, pooling layer compresses information contained within input.

Type of operation that is performed on each element determines a type of pooling layer. This operation can be averaging over units within element, selecting maximal value from element or alternatively learned linear combination of units within element. Learned linear combination introduces form of learning into the pooling layer, but it is not very prevalent.

Selecting of maximal value is most common type of pooling operation and in that case the layer is called Max-Pooling accordingly. Positive effect of Max-pooling

down-sampling is that extracted features that are learned in convolution are invariant to small shift of input. Principle of Max-Pooling is illustrated on Figure 3.6.

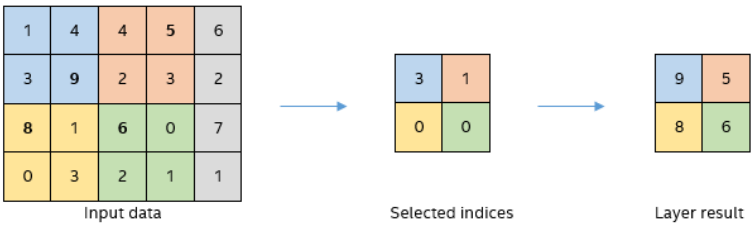

Figure 3.6: Principle of Max-pooling [12]

As already mentioned another advantage of Max-pooling arises when combined with tiled convolution. To create simple detector that is invariant to rotation it possible to use 4 different kernels that are rotated by 90 degrees among each other and when the tiled convolution is used to tile them in groups of 4, the Max-pooling makes sure that resulted feature map contains output from the kernel with strongest signal (i.e. the one trained for that specific rotation of the feature).

Max-Pooling layer will be used to describe process of training of CNNs.

Fully-Connected layer

Fully-Connected layer is identical to layer from Fully Connected Neural Network (FCNN) that was already described. Its training also follows already described process.

3.3.2 Training of CNN

Optimization process of CNN is analogues to FCNN. Situation with CNN is more complicated because network is composed of different types of layers. Forward signal propagation and backward error propagation are following special rules for each layer. Equations used in this section were inspired from [15].

First phase is called forward-propagation, where the signal is propagated from inputs of the CNNs to its output. In the last layer the output is compared with desired value by cost function and error is estimated. In second phase is again used back-propagation algorithm to estimate error contribution of individual units. Variable parameters of the network are again optimization by gradient descent algorithm.

Forward Propagation of Convolution Layer

Each convolutional layer is preforming convolution operation on its input. Presuming that input of a layer is of size $N \times N$ units and kernel is of size $m \times m$. Convolution is computed over $(N-m+1) \times (N-m+1)$ units without zero padding.

Computation of convolution output x_{ij}^l is defined as

$$x_{ij}^l = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \omega_{ab} y_{(i+a)(j+b)}^{l-1} \tag{3.29}$$

where $i, j \in (0, N-m+1)$, l is index of current layer, ω_{ab} are weights of the kernel and $y_{(i+a)(j+b)}^{l-1}$ is output of previous layer.

Output of convolutional layer y_{ij}^l is computed by squashing of output of convolution operation x_{ij}^l through non-linearity:

$$y_{ij}^l = g(x_{ij}^l) \tag{3.30}$$

where g represents this non-linear function.

Backward Propagation of Convolution Layer

Backward propagation for convolutional layer is following the same principles as described in Section 3.2.4. The difference is in fact that convolution kernel shares weights for entire layer and kernels do not have bias described in Section 3.2.1.

Given partial derivative of error from previous layer with respect to output of convolutional layer $\frac{\partial C}{\partial y_{ij}^l}$, influence of kernel weights on the cost function it needs to computed

$$\frac{\partial C}{\partial \omega_{ab}} = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \frac{\partial C}{\partial x_{ij}^l} \frac{\partial x_{ij}^l}{\partial \omega_{ab}}. \tag{3.31}$$

From Equation 3.29 it follows that $\frac{\partial x_{ij}^l}{\partial \omega_{ab}} = y_{(i+a)(j+b)}^{l-1}$, thus

$$\frac{\partial C}{\partial \omega_{ab}} = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \frac{\partial C}{\partial x_{ij}^l} y_{(i+a)(j+b)}^{l-1}. \tag{3.32}$$

To compute deltas (equivalent to Equation 3.9) $\frac{\partial C}{\partial x_{ij}^{(l)}}$ using the chain rule

$$\frac{\partial C}{\partial x_{ij}^{(l)}} = \frac{\partial C}{\partial y_{ij}^l} \frac{\partial y_{ij}^l}{\partial x_{ij}^l} = \frac{\partial C}{\partial y_{ij}^l} \frac{\partial}{\partial x_{ij}^l} \left(g' \left(x_{ij}^l \right) \right) = \frac{\partial C}{\partial y_{ij}^l} g' \left(x_{ij}^l \right) \tag{3.33}$$

Since $\frac{\partial C}{\partial y_{ij}^l}$ is already given the deltas is computed by derivation of activation function. Last step comes to propagation of error into previous layer by equation

$$\frac{\partial C}{\partial y_{ij}^{l-1}} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \frac{\partial C}{\partial x_{(i-a)(j-b)}^l} \frac{\partial x_{(i-a)(j-b)}^l}{\partial y_{ij}^{l-1}} \tag{3.34}$$

Again from Equation 3.29 it follows that $\frac{\partial x^l_{(i-a)(j-b)}}{\partial y^{l-1}_{ij}} = \omega_{ab}$, therefore

$$\frac{\partial C}{\partial y^{l-1}_{ij}} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \frac{\partial C}{\partial x^l_{(i-a)(j-b)}} \omega_{ab}. \tag{3.35}$$

The result looks suspiciously similar to convolution operation and can be interpreted as convolution of error with flipped kernel.

Forward Propagation of Pooling layer

Feed forward operation of pooling layer is strait forward as described in section 3.3.1. Ratio is typically 4 to 1, which means that input matrix is divided into not-overlapping sub-matrices of size 2×2 and each of these produces 1 output. Another possibility is to have overlapping sub-matrices, where size of sub-matrix is larger the number of pixels between application of pooling.

Backward Propagation of Pooling Layer

As mentioned in section for forward-propagation, there is no explicit learning process happening in pooling layer. Error is propagated backwards depending on how the signal was propagated forward. In case of Max-Pooling layer the error is propagated only to the unit with maximal output in forward-propagation phase (in other words to the winner of pooling). The error is propagated very sparsely, as result.

3.4 Regularization of Neural Networks

Control of complexity applies to both NN and CNN. There are several popular regularization techniques that mostly consist of modification of cost function or optimization algorithm. Slightly different approach is to modify structure of the network during training phase.

Dropout

By far the best regularization method is to combine predictions of many different models. This method greatly improves generalization ability of combined model while preventing over-fitting. Exactly on this idea are based ensemble models. The problem with ensemble models is that they are computationally expensive. Because of this, ensembles are usually composed of many very simple models [30].

This idea is especially problematic with DNNs, which are model with many parameters that are difficult to train. Moreover, even when trained models are available in some applications it still isn't feasible to evaluate many different models

in production environment. Another problem is that there might not be enough data to train these different models.

All of these problems can be solved by dropout technique. The basic idea is that each neuron in the network has certain probability to be deactivated during one iteration. This potential for deactivation is evaluated in every iteration, to ensure that network has different architecture every time. Deactivated means that it will not propagate any signal through. This forces individual neurons to learn features that are less dependent on its surrounding.

Probability for deactivation is a hyper-parameter that can be tuned, but reasonable default value is 0.5. Dropping out is only happening in the training phase. In testing phase are all weight connection multiplied by the probability of a dropout. This is done because the activation of the network has to stay roughly equivalent[7] in both training ant testing phase. Basic concept is illustrated in Figure 3.7

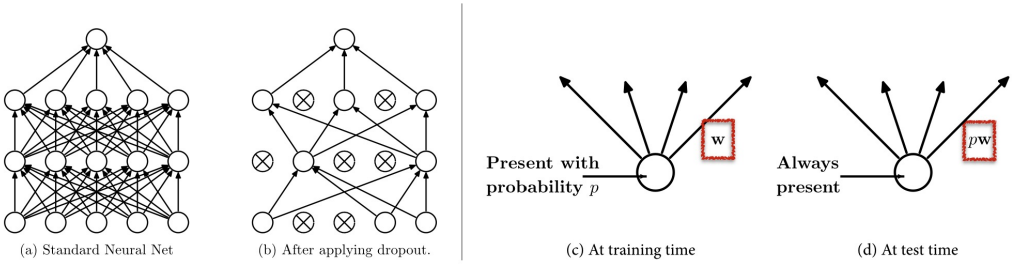

Figure 3.7: Dropout: (a) Standard fully connected network. (b) Network with some neurons deactivated. (c) Activation of neuron during training phase. (d) Activation of neuron during testing phase [4].

[7]For example, when the dropout probability is 0.5, approximately half of the neurons in the network will be deactivated. Therefore in the testing phase the activation would be twice as big.

4 OVERVIEW OF CNN APPLICATIONS

In this chapter is laid out an overview of CNN applications.

4.1 Handwritten Digit Recognition

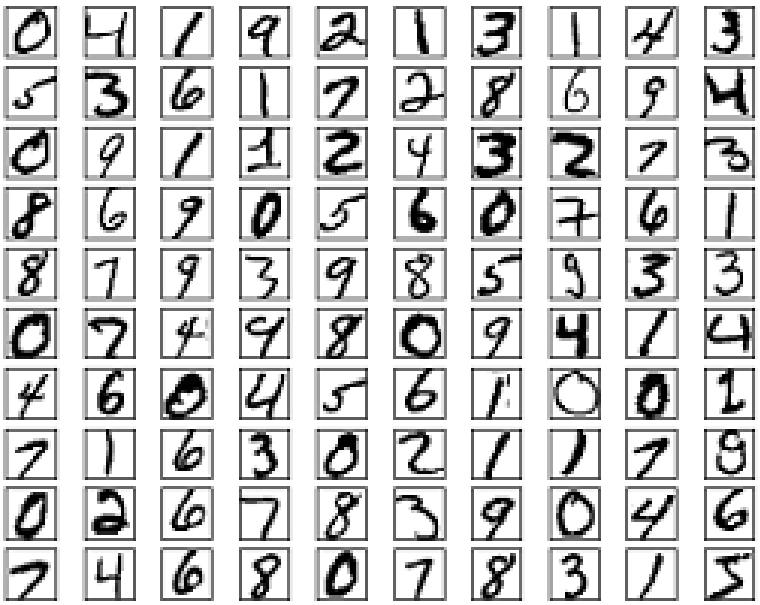

Figure 4.1: Examples of 100 handwritten digits from MNIST dataset [6]

As it was mentioned before CNNs were originally designed for image processing applications. Their invention is credited to Yann LeCun [24]. In 1989 LeCun was working on recognition of hand written digits. Source of the dataset were handwritten zip codes from US post database. Concept was successfully deployed on DSP chip and tested in real-time application of sorting mail by the zip-code. CNN were in this case one of the first models that were capable to replace human operators. Architecture of CNN used to solve this problem is often regarded as LeNet-5.

Modified version of the original dataset was later established as one of the benchmark datasets for image processing. This dataset is called MNIST and it contains 60000 training and 10000 testing gray-scaled images of size 28×28. Example of images from the MNIST dataset are shown on Figure 4.1.

4.2 ImageNet and ILSVRC Competition

ImageNet is a project of Stanford Vision Lab at Stanford University. It is a coordinated effort to gather largest database of annotated images for visual recognition

research. As of writing of this document the database contained 14,197,122 images from 21841 categories. Hierarchy of the ImageNet is meant to map onto WordNet database to cover significant portion of it's nouns.

ImageNet project is probably most well known for its ILSVRC competition happening annually since 2010. Rules of the competition undergo minor updates every year but the main task remains the classification of images into 1000 categories with training dataset of 1.2 million images. These categories cover wide variety of general concepts but it also contains 120 categories for different breeds of dog, which adds problem of fine-grained recognition.

This task is measured on top-1 and top-5 error rates, where top-5 error rate is classified as success in the case that correct label is among first 5 predictions of the model.

What made CNN front and center of everyone's attention was their success in ILSVRC competition in 2012, where model submitted by Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton [23], achieved top-5 test error rate of 15.3%, while the second best performance was 26.2%. This staggering gap in achieved performance garnered large interest of computer vision community. This model is often regarded as AlexNet.

In the following year the competition seen large increase of submitted CNN models. Winners of the 2013 were Matthew Zeiler and Rob Fergus with their ZF Net [33], which had similar structure to AlexNet and achieved 11.2% error rate.

In the 2014 was the ILSVRC dominated by GoogLeNet[1] with top-5 error rate of 6.7%. This model is very different from the previously discussed CNNs and it is using so-called inception modules [31]. Officially it is cited to have 22 layers, but since these layers them self are combination of multiple elements the number is actually over 100.

The year 2015 ILSVRC was won by team from Microsoft with their ResNet model with an error rate of 3.6%. Aside from the actual record in the competition it also broke the record with most layers with 152. Trained network had been so enormous thanks to trick called residual learning [27].

Results of ILSVRC 2016 didn't bring any revolutionary improvements and winner of used ensemble based approach. It is not likely that year 2017 will bring any significant improvement of top-5 error rate from model based on CNN alone, but it remains to be seen [13].

Figure 4.2: Examples of automaticall colorated images [10]

4.3 Colorization of Black and White Images

Automatic colorization is interesting technical problem where the task is to create colored image from gray scale input. Any strides in automatization of this process are welcomed because until recently it has been was very tedious and slow process that needs heavy assistance from human operator. This task also seen some success with regression based models, but resulting images were not very aesthetically pleasing.

Application of very deep CNNs managed to deliver very promising results. In this case convolution network was trained in supervised manner. As inputs, gray scaled images were used. They were trained to categorize detected shapes in gray-scale image to correct color. This technology could be also used to colorize black and white video [35].

4.4 Adding Sounds to Silent Movies

This is very interesting demonstration of capabilities of state of the art Deep Learning models. Solely based on silent video sequence of drumming stick hitting different surfaces with different textures, the model is capable to guess the sound effect that set hitting produces. Convolutional Neural Network was trained to classify the type of surface being hit from visual cues (vibration of hit surface, movement of particles upon impact and so on). And Long Short Term Memory (LSTM) Recurrent neural network was trained to reproduce sound patterns most similar to actual sound that

[1]The name GoogLeNet is a nod to LeCun's model LeNet-5.

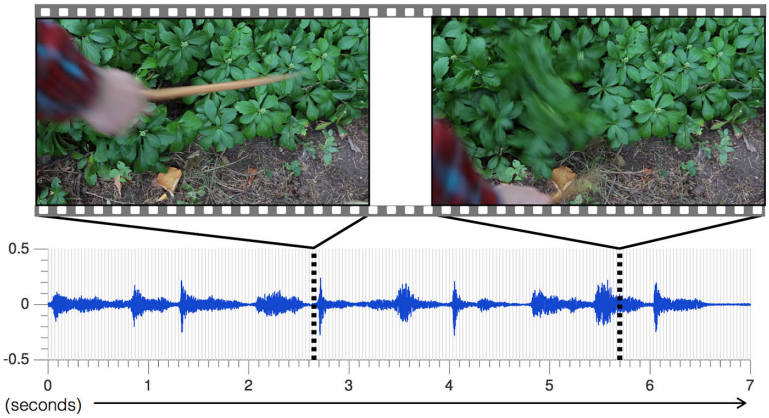

Figure 4.3: Time line of automatically generated sound for silent video [9].

was recorded in original video. Produced sounds were tested with human participants, who had to distinguish synthesized sound from the real ones. Surprisingly in some cases the model fouled to test subjects [27].

4.5 Real-time Machine Translation

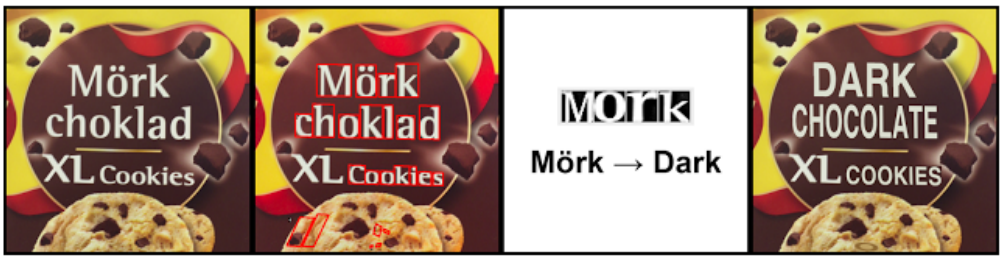

Figure 4.4: Automatic vision translation on image in real time [8].

Convolutional Neural Network was used to detected written text within image scenes and send to large LSTM Recurrent neural network to provide translation. Translated text was then re-rendered back to original image converting foreign text into intelligible (translated). This application was deployed by Google in 2015 to their android devices as extension for Translator application. Feature was called instant visual translation [34].

4.6 Description of Scene in Images

Already familiar combination of CNN and LSTM RNN used in this case to describe scene depicted on image.

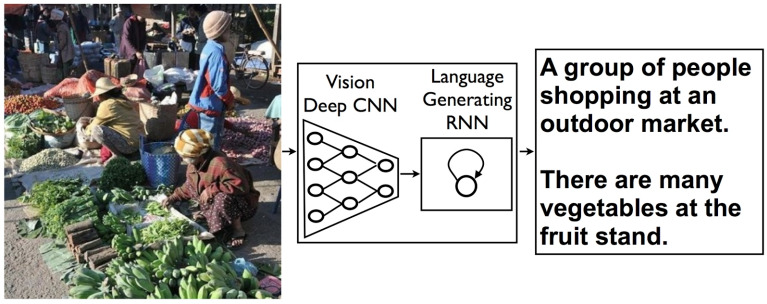

Figure 4.5: Structure of a learning algorithm used for automatic description [3].

Structure of learning algorithm is illustrated on Figure 4.5. CNN was trained do categorize objects on image and last layer of the network was directly fed into language generating LSTM to generate description of the scene.

Figure 4.6 shows examples of translation with varying level of success. It is interesting that the mistakes not really seem outrages and are mostly understandable [32].

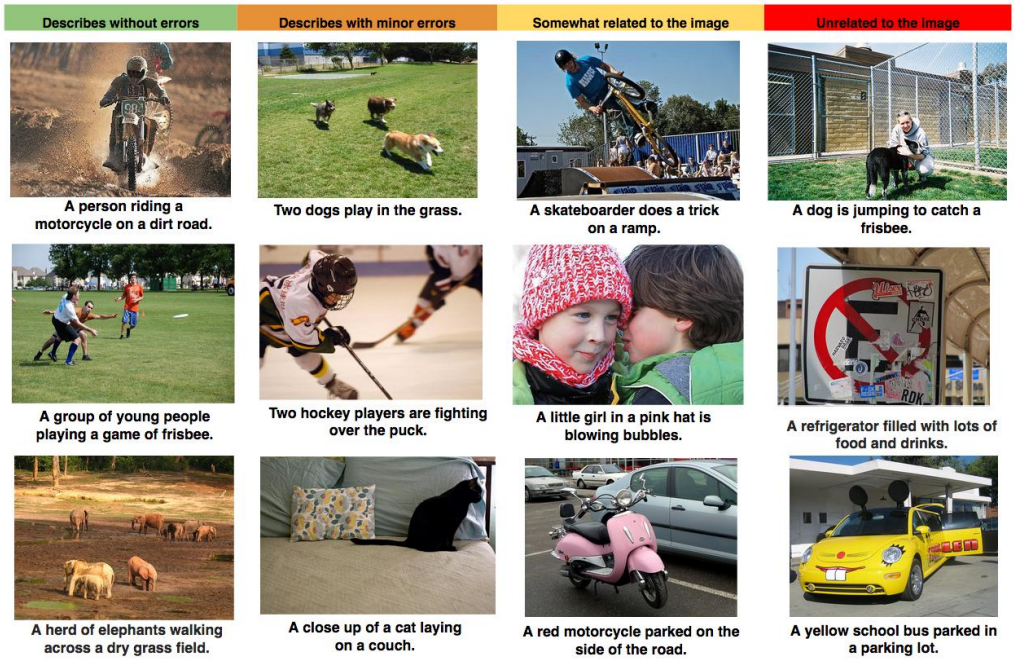

Figure 4.6: Examples of scene description [2].

5 PROPOSED EXPERIMENTAL CONCEPT

This chapter is focused on description of proposed experiment. In the beginning are described available software tools. Following that, is description of software and hardware configuration of testing equipment. Next the preparation of ImageNet dataset is described. The last portion of this chapter is dedicated to specifics of implementation.

5.1 Software Tools

In order to select appropriate tool for implementation of CNN for ILSVRC classification, investigation of available software tools and libraries was conducted. There is vast variety of software tools for machine learning. Some of these are general tools for machine learning, but some are specifically designed for deep learning.

In the last 10, years the software tools for machine learning has undergone a renaissance. There is broad selection of them available and new tools are introduced quite frequently. For example Caffe2[1] was introduced very recently on April 18th. Almost every commonly used programming language has either some software library or at least some available Application Programming Interface (API).

The selection of the software tool was influenced by several factors. Firstly the implementing language had to be well know and somewhat mainstream. Enough of available learning materials had to be available, preferably in form of tutorials. The most important factor was good support for learning on GPU.

Theano

It is seasoned python library. Designed to define, optimize and evaluate mathematical expression with multi-dimensional arrays. This makes it suitable for machine learning needs. Theano is built on top of Numpy, which is python module that enables efficient operation with tensors and basic image processing technique. Combination of Numpy and Scipy brings rich set of tools for image processing and data processing. Its capabilities can arguably rival MatLab, while being open source and free.

Theano's biggest rival is currently Tensorflow project. One of the problems of Theano is its low level nature. Implementation of machine learning algorithms directly can be very complicated. This is probably the reason it slowly falling by the way side. This is also the reason why Theano as a tool is not very suitable for direct implementation of CNN models.

[1]Newer version of popular Caffe framework now available at `https://caffe2.ai/`.

Torch

Torch is one of the oldest frameworks for scientific computing. It is rich and powerful tool that was one the first heavily used for deep learning applications. Similarly to other items in this list it offers fast and efficient GPU support.

Minor negative of Torch is that it uses Lua scripting language as a programming interface. Lua is not very commonly used and as such it suffers from lack of interest of the mainstream machine learning community.

Tensorflow

Tensorflow is very similar to Theano. As the name suggest this library is focused on effective work with tensors. It was originally developed for internal use in Google for machine learning task but it was released as open source in 2015. Tensofrlow computations are expressed as stateful dataflow graphs, which enables efficient support for GPU aided computation. Is currently advertised as one of the fastest frameworks for deep learning needs.

Its disadvantage is similar to Theano, in the fact that it is very low level and direct usage for implementation of Deep learning models is not ideal.

Caffe

Caffe is a deep learning framework that aims to be modular and fast. It is developed by Berkeley AI Research (BAIR) and by community contributors. Yangqing Jia created the project during his PhD at UC Berkeley. It is implemented in C++ but it also offers APIs for several other languages as for example python.

Its biggest drawback is its lack of quality documentation. This fact is partially remedied by the existence of Model Zoo, which is collection of favorite models that are freely available. Caffe was in the last years used by companies as Facebook for example mainly because its performance capabilities. Caffe is more geared towards the development of mass production application than it is for research purposes.

Keras

Keras is relatively young but mature project written in python. It is high lever neural network API. It is build capable of running on top of either Theano or Tensorflow libraries. It is very simple with emphasis on rapid model development. At the same time thanks to python infrastructure it is very easily extensible.

Keras probably currently has one of the largest communities among similar tools for deep learning. It has very good documentation containing many code examples and other resources that help users to get started very quickly.

Since both Theano and Tensorflow support execution models on GPU units it extends this possibility to Keras as well.

5.2 Hardware and Software Configuration

Training of NN is notoriously computational expensive and it demands a lot of resources. From low level perspective it translates into many multiplications of matrices. Modern Central Processing Units (CPUs) are not optimized for such computations and therefore are not very good at it. On the other hand, modern GPUs are designed to preform precisely these operations.

Currently on the market there are two major parallel computing platforms CUDA and OpenCL. They both have pros and cons but the major difference is that CUDA is proprietary, while OpenCL is open source. This divide translates into hardware manufactures as well. CUDA is mainly supported by Nvida and OpenCL is support by AMD. Nvidia with its CUDA platform is currently leader in the domain of deep learning. Therefore, for training of CNN models was was selected GPU from Nvidia. Selected model was GIGABYTE GeForce GTX 1080. Summary of relevant available hardware configuration is in Table 5.1.

Table 5.1: Hardware configuration

GPU	GeForce GTX 1080 8GB
CPU	Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz
Memory	DIMM 1333MHz 8GB

From the list of considered software libraries was selected Keras. The reason being that Keras fulfilled all consideration factors and because it was written in python which was most familiar to the author. Support of efficient GPU in Keras is relying on either Theano or Tensorflow back-end. From the user perspective it doesn't really mater either way, but Tensorflow was selected because it was regarded as faster of the two. Latest supported version of CUDA platform was 7.5. and PC was also equipped with cuDNN v5.1, which is GPU-accelerated library of primitives for deep neural networks. Details of software configuration is summarized in table 5.2.

5.3 Dataset Preparation

Due to a organizational difficulties it took a long time to obtain viable hardware that could have been used for training of the network. It was decided that instead of regular ILSVRC challenge, this thesis will attempted to solve reduced version of

Table 5.2: Software configuration

Keras	2.04
Tensorflow	1.1.0
CUDA	7.5
cuDNN	5.1
Python	3.53
Ubuntu Mate	17.04

the task. This reduction involves number of classes. From original ILSVRC dataset was randomly selected 100 classes. This seamed to be a reasonable compromise to preserve some of the inert difficulty of the problem while meeting the deadline of this thesis. This reduction necessarily influences the predictive value of performed experiment. One of its effects is reduction of the fine-grained recognition present in original dataset. This is due to a fact that original ImageNet dataset contains 120 categories of different breeds of dogs.

Even after the reduction, one of the biggest challenges of the ImageNet dataset, was its size. Typical benchmark datasets for NN applications, MNIST or CIFAR10 are much smaller in comparison. It is feasible to work with them within memory. In case of ImageNet dataset, any augmentation of the dataset that required to load it into memory had to fail[2].

Therefore any data preparation had to be performed in batches. Python ecosystem offers several options for storing of data for mathematical manipulation. For these specific requirements was selected HDF5fn:3 binary data format. Image data were saved into hard-drive using h5py python library.

Images contained in the dataset are quite varied. They have different number of pixels, aspect ratio and overall quality. Selected model imposed the constraint that each image has to be of the same size and aspect ratio. Selected size of the images was 256x256 pixels, based on the work described in [23]. Dataset preparation was done in three stages.

Image Pre-processing

In the first stage was randomly selected 100 categories. Every image from this subset was pre-processed in following way. When both sides of the image were larger than 256 pixels, it was re-sized so that the shorter side matched the 256 pixels and exceeding pixels on the longer side were cropped out. Every image that had one side larger and smaller was filled by zeros on the shorter side and cropped on the longer side. In case that both sides were shorter then 256 pixels it was re-sized in

[2]Reason behind this is that roughly 50000 images in resolution $255 \times 255 \times 3$ in Float32 format requires 36.1 GBytes of memory.

similar way as in case of both sides larger but the size of image was increased instead of decreased.

Split Data into Training a Testing Dataset

In the ILSVRC competition the participants are provided with special validation and testing instances, in this setup the training images were split into two datasets instead. This was done only because it saved time during crucial time period of designing the experiments. This decision has a drawback because it necessarily reduces the amount of data available for training of the network. But since the dataset was substantially smaller then in ILSVRC competition it didn't represent a big problem. Ratio of train to test size was selected 9:1. Training dataset contained 44323 images and testing dataset contained 4925 images.

It was made sure that images are randomly split between test and train dataset so that both dataset have roughly equal split among the categories. It is very important that the model is trained with alternating categories. For example if the model would be exposed to images of the same category in the row the update of weights of in the given cycle would be biased to this category. In other words it would be over fitted, which is not desirable.

It was found that simple random selection of images didn't vouch for equally representation in resulting datasets. This was caused by the fact that not all classes had the same number of training images. Average number of images was around 500 per class, but some classes had over 1000 images. It was solved by intervention on every third draw during random selection. In this intervention was selected the category that currently had most images left.

Format Conversion

In the last stage were images normalized and converted to appropriate data type. Original images were Red Green Blue (RGB) with individual pixels coded as Unit8 type with values in range 0-255. Keras model requires data to be provided in float32 type. Therefore the pre-processed images had to be converted. It was also normalized into range 0-1. To normalize each pixel was simply divided by maximal value of uint8, which is 255.

Data with correct label for the images were integers with index in range 0-100. Keras model expects the label data in categorical format. Categorical format converts each index of the label data into vector of all zeroes but one of length equal to number of categories. Element of the vector with index equal to original index of the labeled data is equal to one.

To convert a instance into categorical format following function was used

```
np_utils.to_categorical(y, num_classes)
```

This function is available in `keras.utils` module.

For illustration in dataset with 5 classes an instance of class label with index 3

```
>>> y_instance
3
>>> np_utils.to_categorical(y_instance, 5)
[0.0, 0.0, 0.0, 1.0, 0.0]
```

Both of these operation needed to be performed on each instance of the data and since this couldn't be done in memory, the whole process was executed in batches of 500 images.

```
# X_train Y_train datasets are devided into batches of 500
for index in range(0, len(X_train), 500):
    index_end = index + 500

    # normalize values
    X_train[index:index_end] = np.divide(
        X_train[index:index_end], 255)

    # convert to categorical
    Y_train[index:index_end] = np_utils.to_categorical(
        Y_train[index:index_end], 100)
```

5.4 Data Augmentation

Main problem with ImageNet dataset is that it has relatively few images per category. This issue was exacerbated by the decision to also use train dataset for testing purposes described in section 5.3. Data augmentation procedure suggested in [23] was performed to combat this. During training of the network each image was augmented before it was fed into the network. Pre-processed images from HDF5 file have size $256 \times 256 \times 3$ pixels. While the size of input of the CNN is setup to process data of size $224 \times 224 \times 3$. Therefore each image that is send on the net's input was randomly generated patch of size $224 \times 224 \times 3$ from the pre-processed image. The generated patch was also flipped horizontally With probability of 0.5. By this augmentation the training dataset is theoretically extended by factor of 2048[3]. Example of the this process is depicted on figure 5.1

[3] *(size of an image side - size of generated patch)*^{2} * *horizontal flip* $[(256 - 224)^2 \times 2 = 2048]$.

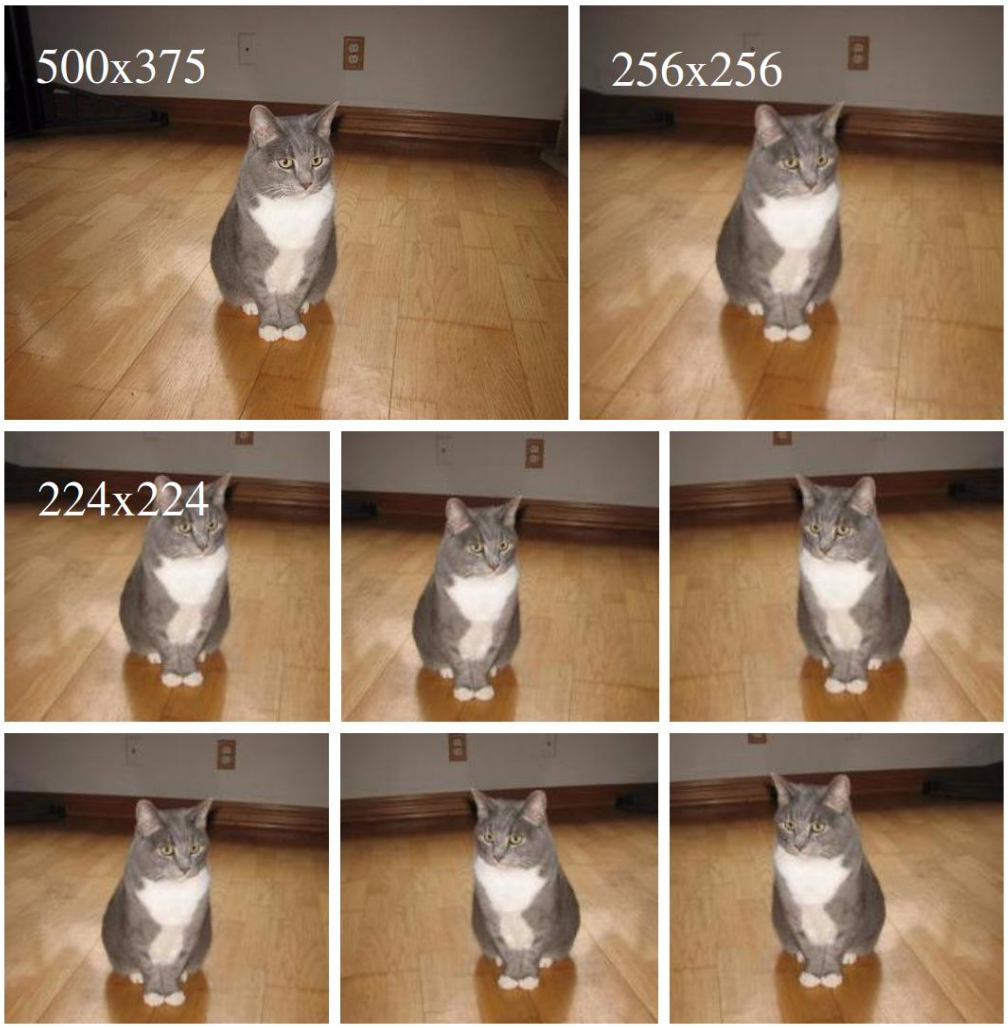

Figure 5.1: Original image (left up). Downsized and cropped on the sides (right up). Six randomly generated patches from processed image (bottom).

5.5 Model Building Blocks

For implementation of CNN was used Keras sequential model, which is a concept that is appropriate for modeling of feed forward network. Definition of the network is composed of layers. Concept of layer in Keras sequential model doesn't completely map into already described definition of layer from topological perspective. Keras layers are more fine grained and in order to create equivalent topological layer it is necessary to use multiple Keras layers.

Model is created simply by calling `Sequential` constructor

```
model = Sequential().
```

Layers are added by calling an `add` method on object of sequential model

```
model.add(layer),
```

where `layer` is definition of the layer.

5.5.1 Keras Layers

All models were created by composition of following layers.

Convolutional

Convolutional layer used in the architecture was of following structure

```
Conv2D(filters=n, kernel_size=(z, z), strides=(s, s), padding='valid',
       input_shape=shape)
```

where `n` is number of filters that the layer will have, z is size of kernel, s is number of pixels in stride and `input_shape` defines size of input matrix.

Activation

To add activation function on the output of the layer user can specify parameter `activation` of the layer itself or create activation as a layer

```
Activation(acitvation_function)
```

where `activation_function` is either 'softmax' or 'relu'. Both specifications are equivalent because Keras automatically uses linear activation function for each layer.

Pooling

Pooling layer can be specified as

```
MaxPooling2D(pool_size=(z, z), strides=(s, s))
```

where `pool_size` specifies size of pooling kernel and `strides` specifies number of pixels in x and y direction that are traversed in between application of individual pools.

Fully Connected

Fully connected layer is created by

```
Dense(num_of_units)
```

where `num_of_units` is a number of fully connected neurons in one layer.

Dropout

Similarly to activation function to apply a dropout regularization on a layer it needs to be added after it as another layer.

```
Dropout(p)
```

where `p` is both probability that any unit is dropped and also the coefficient by which are the outputs multiplied during forward evaluation.

Other

Feature extraction layers are multidimensional. Specifically both Convolutional and Pooling layers are two dimensional. Classification layers that are created by fully connected layers are one dimensional. To connect the two, it is necessary to create mapping between them. For this purposes it necessary to use following layer

```
Flatten()
```

which takes care of necessary connections between layers.

5.5.2 Model Compilation

When the structure of the model is specified, before it can be trained it also needs to have cost function, optimization procedure and metrics defined. This is done be calling `compile` method on the model

```
model.compile(
    loss= 'categorical_crossentropy',
    optimizer=Adam(lr=0.001, beta_1=0.9, beta_2=0.999,
                   epsilon=1e-08, decay=0.0),
    metrics=['accuracy'])
```

parameter `loss` specifies cost function, `optimizer` optimization procedure and `metrics` specifies metrics by which the model is measured.

5.5.3 Model Fitting

Process of model training is in Keras called model fitting. It offers two types of fitting methods `fit` and `fit_generator`. Method `fit` loads entire dataset at once and use it train the network. For reasons laid out earlier this not possible with ImageNet dataset. Second option `fit_generator` method uses feature of python language called generator[4].

Generator `generate_data` was implemented to supply data from ImageNet dataset.

```
def generate_data(hdf5_file_name, batch_size, data_type):
    """Generator that is providing data from dataset in
    infinite loop.
    """
    with h5py.File(hf5_file_name, 'r') as hf5:
        data_x = hf5["/data/%s/x" % data_type]
        data_y = hf5["/data/%s/y" % data_type]
        pos = 0
        size = data_x.shape[0]

        # infinite loop
        while True:
            if pos + step <= size:
                batch_x = data_x[pos:pos + step]
                batch_y = data_y[pos:pos + step]
            else:
                temp = pos
                pos = (pos + step) - size
                batch_x = np.concatenate(
                    (data_x[0:pos], data_x[temp:size]))
                batch_y = np.concatenate(
```

[4]Details about python generators can be found in official documentation.

```
                (data_y[0:pos], data_y[temp:size]))

        pos += step

        augmented_batch_x = np.empty((step, 224, 224, 3))
        for index, image in enumerate(batch_x):
            if data_type == "train":
                augmented_batch_x[index] = \
                    generate_random_patch(image)
            elif data_type == "test":
                augmented_batch_x[index] = \
                    get_center_patch(image)

        yield (augmented_batch_x, batch_y)
```

This generator is generating data from HDF5 file in infinite loop. Depending on the parameter `data_type` it ether generates training or testing data. Difference between the two is that testing data are generated as center patch of size 224x224x3 from the original sized image 256x256x3. Training data are generated as patch of size 224x224x3 from random position. Parameter `batch_size` determines size of generated batch.

Very useful property of fit `fit_generator` method is that while the model is trained on GPU it is able to prepare another batches of training data in parallel. This is very useful because the data augmentation performed by the generator does not slow down the training process.

```
model.fit_generator(
    generator=generate_data(HDF5_FILE_NAME,
                            train_batch_size,
                            "train"),
    steps_per_epoch=steps_per_epoch,
    epochs=epochs,
    verbose=1,
    validation_data=generate_data(HDF5_FILE_NAME,
                                  test_batch_size,
                                  "test"),
    validation_steps=validation_steps,
    max_q_size=10,
    workers=4,
    pickle_safe=True)
```

Method `fit_generator` has been used to train model with following parameters:

- `generator` : initialized with `generate_data` for training data.
- `steps_per_epoch` : number of calls to `generate_data` per each epoch
- `epochs` : number of epoch
- `validation_data` : initialized with `generate_data` for testing data.
- `validation_steps` : number of calls to `generate_data` per testing
- `max_q_size` : number of concurrent generators
- `workers` : number of threads serving the generators

6 EXPERIMENTS

This chapter is broke down into two parts. In first part is described the process of selection of model structure and its parameters. In the second part are summarized achieved results on ImageNet dataset.

6.1 Selection of Model Structure and Parameters

Selection of a NN structure is very difficult problem. Networks that were successfully deployed in large scale applications as for example ILSVRC can have millions of parameters and their training necessitates selection of multiple meta-parameters. All of it put together creates enormous complexity. In order to find appropriate starting point, devised experiments were inspired by structure of AlexNet model . AlexNet model is composed of 5 convolution layers with 1376 kernels and 3 fully connected layers with 9192 neurons. Due to the reduction of the trained classes opposed to ILSVRC, numbers of kernels and neurons was scaled down. At the same time the structure of network was simplified because entire model could be trained on only one GPU card instead of two.

During the experimentation was performed multiple test with differing structure of the network. Most accurate structure that was found was used to further experiment with meta-parameter settings.

6.1.1 Structure of Model

First element that had to be selected was learning algorithm. Even though optimization of AlexNet was performed with SGD algorithm, recent literature [14] suggest that Adam and Nadam optimizer are more effective. For the first approximation was selected Adam, because it has fewer modifiable parameters and therefore, it is more likely to perform well with default parameters.

To find appropriate number of convolution layers it was experimented with 4 and 5. Number of neurons in fully connected layers in first iteration was set 1124. Number of kernels for 4 convolution layers was set to 544 and 271. For 5 convolution layers it was set to 734 and 366. Consequently the same settings were used with 2148 neurons in fully connected layer. Last experiment was performed on net with 5 convolutional layers, 734 and 366 kernels, and 4196 neurons in fully connected layer. All tested combinations are summarized in Table 6.1. Column Accuracy shows maximal accuracy that was achieved in 150 training epochs and column Epoch shows in which epoch it was.

Table 6.1: Influence of network structure on its accuracy.

Conv. Layers	Kernels	Dense Layers	Neurons	Epoch	Accuracy
4	271	3	1124	132	0.5844
4	544	3	2148	120	0.5833
4	544	3	1124	132	0.5805
5	734	3	1124	148	0.5694
5	734	3	2148	132	0.5688
5	366	3	2148	125	0.5683
5	366	3	4196	145	0.5622
5	734	3	4196	131	0.5461
5	366	3	1124	147	0.5383
4	271	3	2148	88	0.5288

Structure with highest achieved accuracy was structure with 4 convolutional layers, 271 kernels, 3 fully connected layers and 1124 neurons. This structure was also tested with Nadam and SGD optimizer to determined best option for this structure. Training was performed with default parameters that are predefined in Keras library. Neither Nadam nor SGD brought any improvements in accuracy. It is interesting to compare original structure of AlexNet with the selected. ILSVRC dataset was reduced approximately by factor of 10 and number of convolution kernels is 5 smaller and number of neurons in fully connected layer is 8 times smaller.

6.1.2 Parameters of Learning Algorithm

With selected structure and learning algorithm it was further experimented to determine optimal configuration of its parameters. In this experiment were modified following parameters:

- learning rate (0.0001, 0.00015, 0.0002)
- beta 1 (0.8, 0.85, 0.9)
- beta 2 (0.995, 0.997, 0.999)

Best performance was achieved with combination:

- learning rate = 0.0002;
- beta 1 = 0.8;
- beta 2 = 0.999.

Summary of all results is in Table 6.2. It was observed that biggest influence was caused by change of learning rate. This suggested that it requires finer tuning. Table 6.3 summarizes achieved accuracy for different settings of learning rate. Best performance was achieved with learning rate set to 0.00008.

Table 6.2: Accuracy of different learning parameters.

Learning Rate	Beta 1	Beta 2	Epoch	Accuracy
0.0001	0.8	0.995	9	0.37
0.0001	0.8	0.997	8	0.38
0.0001	0.8	0.999	7	0.38
0.0001	0.85	0.995	9	0.35
0.0001	0.85	0.997	8	0.36
0.0001	0.85	0.999	8	0.36
0.0001	0.9	0.995	9	0.36
0.0001	0.9	0.997	9	0.34
0.0001	0.9	0.999	8	0.34
0.00015	0.8	0.995	8	0.40
0.00015	0.8	0.997	8	0.35
0.00015	0.8	0.999	7	0.39
0.00015	0.85	0.995	8	0.35
0.00015	0.85	0.997	9	0.36
0.00015	0.85	0.999	8	0.34
0.00015	0.9	0.995	9	0.34
0.00015	0.9	0.997	9	0.36
0.00015	0.9	0.999	8	0.35
0.0002	0.8	0.995	7	0.35
0.0002	0.8	0.997	8	0.37
0.0002	0.8	0.999	8	0.40
0.0002	0.85	0.995	8	0.38
0.0002	0.85	0.997	9	0.37
0.0002	0.85	0.999	8	0.32
0.0002	0.9	0.995	8	0.33
0.0002	0.9	0.997	8	0.30
0.0002	0.9	0.999	9	0.31

Table 6.3: Summary of Learning rate influence on model accuracy.

learning rate	Accuracy	Epoch
0.00008	0.53	59
0.00010	0.51	50
0.00012	0.53	59
0.00014	0.50	39
0.00016	0.49	54
0.00018	0.51	30
0.00020	0.46	39
0.01000	0.05	00

Weight Decay

According to Krizhevsky the key parameter that allowed the CNN to train successfully was weight decay. This unfortunately didn't seem to hold true when using Adam optimizer. Several experiments with weight decay varying from 0 to 0.0005 were performed and in all cases other then decay set to 0 the testing error was manifesting over-fitting.

Table 6.4: Summary of Weight Decay influence on model accuracy.

Weight Decay	Epoch	Accuracy
0.0001	139	0.53
0.0003	128	0.47
0.0005	138	0.43

Dropout

Suggestion from Krizhevsky's paper was to use dropout regularization only in first and second fully connected layers. During the experimentation it was also tried to add dropout in-between the convolution layers. This didn't bring any improvements in performance. Influence of change of a dropout probability was also investigated, but the optimal settings was found to be 0.5.

6.1.3 Batch Size

Last parameter that was taken into consideration was size of training batch. In order to compare accuracy of training with differing batch size the number of total images was 500 000. This means that for batch size 0f 600 images it took 833 batches and so on. Summary of tested batch size is in Table 6.5. Based on the accuracy alone it suggest that best batch size is 600 images per epoch, the problem is that training takes almost 4 times as long. This is caused by the fact that in each epoch the training algorithm has to go through entire testing dataset.

Table 6.5: Summary of Batch Size influence on model accuracy and training time.

Batch Size	Total Batches	Accuracy	Training time [s]
600	833	0.52	8268
1000	500	0.52	6242
2000	250	0.51	4690
6000	83	0.50	3705
10000	50	0.50	2425

6.2 Results

Structure of final CNN model is summarized in Table 6.6. Network was trained using Adam learning algorithm with following parameters:

- Learning rate: 0.00008,
- Beta 1: 0.8
- Beta 2: 0.999
- Weight Decay: 0.0

Size of training batch was 10000 images.

Table 6.6: Structure of model with highest achieved accuracy.

layer	name	kernels	size of kernel	activation function	regularization
1	Conv2D	48	(11, 11)	relu	-
2	MaxPooling2D	-	(3, 3)	-	-
3	Conv2D	64	(5, 5)	relu	-
4	MaxPooling2D	-	(3, 3)	-	-
5	Conv2D	95	(3, 3)	relu	-
6	Conv2D	64	(3, 3)	relu	-
7	MaxPooling2D	-	(3, 3)	-	-
layer	name	neurons	-	activation function	regularization
8	Dense	512	-	relu	Dropout(0.5)
9	Dense	512	-	relu	Dropout(0.5)
10	Dense	100	-	soft max	-

Training of final configuration is illustrated in Figure 6.1. Best noted accuracy for top-1 guess was estimated in epoch 250, with training error of 0.8389 and testing error 0.6156. This model was also evaluated for accuracy of top-3 0.7804 and top-5 0.8474 guesses. Comparison of model accuracy with other CNN models from previous ILSVRC competition is summarized in Table 6.7. Examples of the network can be found on Figures 6.2, 6.3 and 6.4.

Table 6.7: Comparison of implemented model with ILSVRC competition submissions.

	Top-1 Accuracy	Top-5 Accuracy
implementation	0.62	0.84
AlexNet	0.63	0.83
GoogLeNet	-	0.93
ResNet	0.78	0.94

Given the fact that for this experiment was used reduced dataset with only 100 classes, it was expected that the accuracy of model will be noticeable better accuracy of AlexNet. In actuality top-1 accuracy is slightly worse and top-5 only slightly better. This can be caused by variety of factors. Configuration of CNN parameters is daunting task and there is always space for improvement. For example some

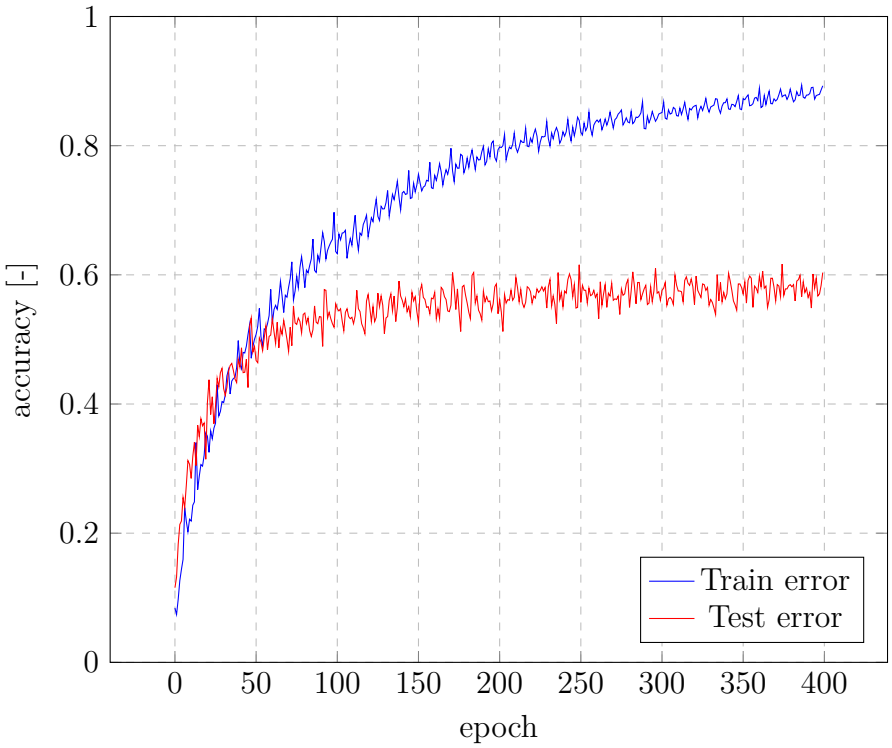

Figure 6.1: Test of final configuration and parameters.

literature suggests that nadam learning algorithm can achieve better performance [14]. This was not found during limited time of its testing it.

Another possible reason behind inferior performance can be insufficient data augmentation. Article by Krizhevsky suggests augmentation by random adjustment of image brightness. Already implemented augmentation could be also expanded by geometric transformation of the images (e.g. rotation).

(a) class = forg

(b) class = piano, prediction = piano

(c) class = dog

(d) class = harmonica

Figure 6.2: Correctly classified images

(a) class = rabbit, prediction = pig

(b) class = dog, prediction = group of people

(c) c = cow, p = dog

(d) class = jellyfixh, prediction frog

Figure 6.3: Guess of Images in Top-5

(a) class = dog, prediction = frog

(b) class = fish, prediction = cow

(c) class = snake, prediction = harmonica

(d) class = monior, prediction = frog

Figure 6.4: Incorrectly Classified Images

7 CONCLUSION

This thesis described brief history of Neural Networks, and its evolution into deep learning models. It highlighted differences of Convolutional and Fully Connected Neural Networks. Several examples of state of the art applications of Convolutional Neural Network were described.

Practical experiments with Convolutional Neural Network models were proposed and. Proposed models were implemented using Keras library with support of Tensorflow. Training of implemented models was aided by parallel computing platform CUDA on Gforce GTX 1080 GPU.

Trained model was compared to latest results of ILSVRC competition. Somewhat surprisingly, best achieved top-1 accuracy of 0.6156 did not surpassed the AlexNet's top-1 accuracy of 0.625 on original ILSVRC dataset. This was surprising because reduced dataset had one-tenth of it's original classes. Possible explanation is that reduced dataset was too small and caused over-fitting of selected model.

Future work would include expansion of selected dataset to full extend of ILSVRC. Another interesting avenue seems to be in non-conventional structure of Convolutional Neural Network. This includes inception modules from GoogLeNet or residual learning from Microsoft's ResNet.

BIBLIOGRAPHY

[1] Over under fitting. Knewton blog developer blog [online]; https://18784-presscdn-0-49-pagely.netdna-ssl.com/wp-content/uploads/2014/09/Gizem1.jpg.png, 2014 (accessed May 13, 2017).

[2] Scene description. Imaging Resource [online]; http://www.imaging-resource.com/?ACT=44&fid=17&d=4247&f=google-image-recognition-2.jpg, 2014 (accessed May 13, 2017).

[3] Algorithm for image description. Semantic Scholar [online]; https://ai2-s2-public.s3.amazonaws.com/figures/2016-11-08/11da2d589485685f792a8ac79d4c2e589e5f77bd/0-Figure1-1.png, 2015 (accessed May 13, 2017).

[4] Dropout. http://lamda.nju.edu.cn/weixs/project/CNNTricks/imgs/dropout.png, 2015 (accessed May 13, 2017).

[5] An image of a traffic sign is filtered by 4 5×5 convolutional kernels. Nvidia Developer; https://devblogs.nvidia.com/parallelforall/wp-content/uploads/2015/11/fig1.png, 2015 (accessed May 13, 2017).

[6] Mnits 100 digits. Neural Networks and Deep Learning [online]; http://neuralnetworksanddeeplearning.com/images/mnist_100_digits.png, 2015 (accessed May 13, 2017).

[7] Test vs. training error. Stack Exange [online] https://i.stack.imgur.com/IpI8U.png, 2015 (accessed May 13, 2017).

[8] Visual translation. ZDnet [online]; http://zdnet3.cbsistatic.com/hub/i/r/2015/07/30/5d3f769d-781c-469e-a487-135529cf1f75/resize/770xauto/55a2022325bcf30b7ee18466f340f195/screen-shot-2015-07-30-at-13-33-07.png, 2015 (accessed May 13, 2017).

[9] Adding sound. Massachusetts Institut of Technology News [online]; http://news.mit.edu/sites/mit.edu.newsoffice/files/styles/news_article_image_top_slideshow/public/images/2016/MIT-CSAIL-sound-prediction-algorithm-2_0.png?itok=LIsXEB-x, 2016 (accessed May 13, 2017).

[10] Colorization. Grossum [online]; https://www.grossum.com/uploads/media/postPicture/0001/01/815d91e877cc1e586702a04ea8837fd8a846367b.jpeg, 2016 (accessed May 13, 2017).

[11] A zero-padded 4 x 4 matrix becomes a 6 x 6 matrix. XRDS Crossroads the ACM Magazine for Students; http://xrds.acm.org/blog/wp-content/uploads/2016/06/Figure_3.png, 2016 (accessed May 13, 2017).

[12] Max pooling. Intel Developer Zone [online]; https://software.intel.com/sites/default/files/did_feeds_images/46c3bfae-84f5-48ed-9412-8c8e9a4df219/46c3bfae-84f5-48ed-9412-8c8e9a4df219-imageId=542924f6-128a-4e45-8c09-b2438e2faec7.png, 2017 (accessed May 13, 2017).

[13] A. Deshpande. The 9 deep learning papers you need to know about (understanding cnns part 3), August 2016 (accessed May 6, 2017).

[14] T. Dozat. Incorporating nesterov momentum into adam. 2015.

[15] A. Gibiansky. Convolutional neural networks, February 2014 (accessed May 12, 2017).

[16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press.

[17] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer New York Inc., New York, NY, USA, 2 edition, 2008.

[18] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[19] D. H. Hubel and T. N. Wiesel. Receptive fields of single neurons in the cat's striate cortex. *Journal of Physiology*, 148:574–591, 1959.

[20] I. Kalová. Předzpracování obrazu. VUT Brno Scriptum, Pocitacove videni, Computer Vision Group, 2015 (accessed May 13, 2017).

[21] A. Karpathy. Cs231n: Convolutional neural networks for visual recognition. http://cs231n.github.io/convolutional-networks/, 2017 (accessed May 13, 2017).

[22] T. Kohonen, M. R. Schroeder, and T. S. Huang, editors. *Self-Organizing Maps*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 3rd edition, 2001.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. pages 1106–1114, 2012.

[24] Y. LeCun, L. D. Jackel, B. Boser, J. S. Denker, H. P. Graf, I. Guyon, D. Henderson, R. E. Howard, and W. Hubbard. Handwritten digit recognition: Applications of neural net chips and automatic learning. *IEEE Communication*, pages 41–46, November 1989. invited paper.

[25] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent nervous activity. *Bulletin of Mathematical Boiphysics*, pages 115–133, 1943.

[26] M. Minsky and S. Papert. *Perceptrons: an introduction to computational geometry*. MIT Press, 1969.

[27] A. Owens, P. Isola, J. H. McDermott, A. Torralba, E. H. Adelson, and W. T. Freeman. Visually indicated sounds. *CoRR*, abs/1512.08512, 2015.

[28] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

[29] T. Sadhu. Machine learning: Introduction to the artificial neural network. `http://durofy.com/machine-learning-introduction-to-the-artificial-neural-network/`, 2012 (accessed May 13, 2017).

[30] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[31] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[32] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. *CoRR*, abs/1411.4555, 2014.

[33] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.

[34] J. Zhang and C. Zong. pages 16–25.

[35] R. Zhang, P. Isola, and A. A. Efros. Colorful image colorization. 2016.

LIST OF TERMS

activation function Function that usually performes non-linear transformation of inputs. 22–24, 29, 31

adam Learning algorithm based on gradient descent [14]. 51, 55

back-propagation Technique that enables the computation of error distribution throughout the network. 19, 24, 25, 30

categorical cross entropy Cost function used for neural networks with classification into multiple classes. 24

classification Machine learning task that tries to assign correct class to a data instance. 10, 11, 19, 24

cost function Equivalent to loss function. 15–17, 24, 30–32

deep learning Popularized term that refers to machine learning models that have architecture that contains many layers. 20, 22, 39

detector stage Application of activation function. 29

ensemble model special case of models that is using prediction of multiple different models with the same input data in order to achieve higher accuracy. 19, 32

feature map Output of convolution operation over input. 28, 30

fully connected Outputs of each neuron in l layer are connected into inputs of all neurons in $l + 1$ layer. 27

gradient descent Learning algorithm based on following path of steepest negative change in gradient. 25, 30

hidden layer Referring to a layer that is neither input nor output, but merely somewhere between. 20

hopfield network Specific type of recurrent neural network invented by John Hopfield in 1982. 19

hyperbolic tangent Activation function commonly used in recurrent neural networks. 22

K-means Unsupervised learning algorithm. 13

kernel Filter used by convolutional layer. 28, 30, 31

linear regression Linear model. 14–16, 21

locally connected Convolution layer that has kernel of the same size as input. 28

loss function Function that is used to estimate error of a model on given data. 15

machine learning research study dealing with models able to learn from experience. 10–12

Max-Pooling Method of input size reduction in convolutional neural networks. 29, 30, 32

nadam Learning algorithm based on gradient descent [14]. 51, 52, 56

over fitting Statistical term describing models inferior ability to capture underlying relationship between input and output due to an abundance of complexity. 16

Perceptron Model of artificial neuron. 19

regression Machine learning task for approximation of continuous-valued function response. 11

sigmoid Activation function historically used in Perceptron model. 22, 23

softmax Activation function commonly used output layer of feed forward neural networks. 24

stride It specifies size of convolution step. Default value is one, which means that kernel is moved by one pixel in each direction. 28

sum of square Loss function. 15

tiled convolution Convolution layer with differing coverage in neighboring pixels. 28–30

under fitting Statistical term describing models inferior ability to capture underlying relationship between input and output due to a lack of complexity. 16

zero padding Expansion of input data by adding elements of zero value around the edges. 28

ACRONYMS

AI Artificial Intelligence. 10, 19, 26
API Application Programming Interface. 39, 40

CNN Convolutional Neural Network. 9, 19, 20, 26–30, 32, 34–39, 41, 44, 46, 54, 55
CPU Central Processing Unit. 41
CT Computed Tomography. 27

DNN Deep Neural Network. 20, 32
DSP Digital Signal Processor. 34

FCNN Fully Connected Neural Network. 30

GPU Graphics Processing Unit. 20, 39–41, 49, 51, 60

HDF5 Hierarchical Data Format 5. 42, 44

ILSVRC Large Scale Visual Recognition Challenge. 9, 35, 39, 41–43, 51, 55, 60

KNN K-nearest Neighbors. 12

LSTM Long Short Term Memory. 36–38

MLP Multilayer Perceptron. 19

NN Neural Network. 14, 19–26, 29, 32, 41, 42, 51

PCA Principal Component Analysis. 13

ReLU Restricted Linear Unit. 22, 23, 29
RGB Red Green Blue. 43
RNN Recurrent Neural Network. 19, 22, 37

SGD Stochastic Gradient Descent. 26, 51, 52
SOM Self Organizing Map. 13, 19
SVM Support Vector Machines. 12, 19

LIST OF APPENDICES

A APPENDIX

Table A.1: Structure of model 4 cl full size 1124 fc

layer	name	kernels	size of kernel	activation function	regularization
1	Conv2D	96	(11, 11)	relu	-
2	MaxPooling2D	-	(3, 3)	-	-
3	Conv2D	128	(5, 5)	relu	-
4	MaxPooling2D	-	(3, 3)	-	-
5	Conv2D	190	(3, 3)	relu	-
6	Conv2D	128	(3, 3)	relu	-
7	MaxPooling2D	-	(3, 3)	-	-
layer	name	neurons	-	activation function	regularization
8	Dense	512	-	relu	Dropout(0.5)
9	Dense	512	-	relu	Dropout(0.5)
10	Dense	100	-	softmax	-

Table A.2: Structure of model 4 cl full size 2148 fc

layer	name	kernels	size of kernel	activation function	regularization
1	Conv2D	96	(11, 11)	relu	-
2	MaxPooling2D	-	(3, 3)	-	-
3	Conv2D	128	(5, 5)	relu	-
4	MaxPooling2D	-	(3, 3)	-	-
5	Conv2D	190	(3, 3)	relu	-
6	Conv2D	128	(3, 3)	relu	-
7	MaxPooling2D	-	(3, 3)	-	-
layer	name	neurons	-	activation function	regularization
8	Dense	1024	-	relu	Dropout(0.5)
9	Dense	1024	-	relu	Dropout(0.5)
10	Dense	100	-	softmax	-

Table A.3: Structure of model 4 cl half size 1124 fc

layer	name	kernels	size of kernel	activation function	regularization
1	Conv2D	48	(11, 11)	relu	-
2	MaxPooling2D	-	(3, 3)	-	-
3	Conv2D	64	(5, 5)	relu	-
4	MaxPooling2D	-	(3, 3)	-	-
5	Conv2D	95	(3, 3)	relu	-
6	Conv2D	64	(3, 3)	relu	-
7	MaxPooling2D	-	(3, 3)	-	-
layer	name	neurons	-	activation function	regularization
8	Dense	512	-	relu	Dropout(0.5)
9	Dense	512	-	relu	Dropout(0.5)
10	Dense	100	-	softmax	-

Table A.4: Structure of model 4 cl half size 2148 fc

layer	name	kernels	size of kernel	activation function	regularization
1	Conv2D	48	(11, 11)	relu	-
2	MaxPooling2D	-	(3, 3)	-	-
3	Conv2D	64	(5, 5)	relu	-
4	MaxPooling2D	-	(3, 3)	-	-
5	Conv2D	95	(3, 3)	relu	-
6	Conv2D	64	(3, 3)	relu	-
7	MaxPooling2D	-	(3, 3)	-	-
layer	name	neurons	-	activation function	regularization
8	Dense	1024	-	relu	Dropout(0.5)
9	Dense	1024	-	relu	Dropout(0.5)
10	Dense	100	-	softmax	-

Table A.5: Structure of model 5 cl full size 1124 fc

layer	name	kernels	size of kernel	activation function	regularization
1	Conv2D	96	(11, 11)	relu	-
2	MaxPooling2D	-	(3, 3)	-	-
3	Conv2D	128	(5, 5)	relu	-
4	MaxPooling2D	-	(3, 3)	-	-
5	Conv2D	190	(3, 3)	relu	-
6	Conv2D	190	(3, 3)	relu	-
7	Conv2D	128	(3, 3)	relu	-
8	MaxPooling2D	-	(3, 3)	-	-
layer	name	neurons	-	activation function	regularization
9	Dense	512	-	relu	Dropout(0.5)
10	Dense	512	-	relu	Dropout(0.5)
11	Dense	100	-	softmax	-

Table A.6: Structure of model 5 cl full size 2148 fc

layer	name	kernels	size of kernel	activation function	regularization
1	Conv2D	96	(11, 11)	relu	-
2	MaxPooling2D	-	(3, 3)	-	-
3	Conv2D	128	(5, 5)	relu	-
4	MaxPooling2D	-	(3, 3)	-	-
5	Conv2D	190	(3, 3)	relu	-
6	Conv2D	190	(3, 3)	relu	-
7	Conv2D	128	(3, 3)	relu	-
8	MaxPooling2D	-	(3, 3)	-	-
layer	name	neurons	-	activation function	regularization
9	Dense	1024	-	relu	Dropout(0.5)
10	Dense	1024	-	relu	Dropout(0.5)
11	Dense	100	-	softmax	-

Table A.7: Structure of model 5 cl full size 4196 fc

layer	name	kernels	size of kernel	activation function	regularization
1	Conv2D	96	(11, 11)	relu	-
2	MaxPooling2D	-	(3, 3)	-	-
3	Conv2D	128	(5, 5)	relu	-
4	MaxPooling2D	-	(3, 3)	-	-
5	Conv2D	190	(3, 3)	relu	-
6	Conv2D	190	(3, 3)	relu	-
7	Conv2D	128	(3, 3)	relu	-
8	MaxPooling2D	-	(3, 3)	-	-
layer	name	neurons	-	activation function	regularization
9	Dense	2024	-	relu	Dropout(0.5)
10	Dense	2024	-	relu	Dropout(0.5)
11	Dense	100	-	softmax	-

Table A.8: Structure of model 5 cl half size 1124 fc

layer	name	kernels	size of kernel	activation function	regularization
1	Conv2D	48	(11, 11)	relu	-
2	MaxPooling2D	-	(3, 3)	-	-
3	Conv2D	64	(5, 5)	relu	-
4	MaxPooling2D	-	(3, 3)	-	-
5	Conv2D	95	(3, 3)	relu	-
6	Conv2D	95	(3, 3)	relu	-
7	Conv2D	64	(3, 3)	relu	-
8	MaxPooling2D	-	(3, 3)	-	-
layer	name	neurons	-	activation function	regularization
9	Dense	512	-	relu	Dropout(0.5)
10	Dense	512	-	relu	Dropout(0.5)
11	Dense	100	-	softmax	-

Table A.9: Structure of model 5 cl half size 2148 fc

layer	name	kernels	size of kernel	activation function	regularization
1	Conv2D	48	(11, 11)	relu	-
2	MaxPooling2D	-	(3, 3)	-	-
3	Conv2D	64	(5, 5)	relu	-
4	MaxPooling2D	-	(3, 3)	-	-
5	Conv2D	95	(3, 3)	relu	-
6	Conv2D	95	(3, 3)	relu	-
7	Conv2D	64	(3, 3)	relu	-
8	MaxPooling2D	-	(3, 3)	-	-
layer	name	neurons	-	activation function	regularization
9	Dense	1024	-	relu	Dropout(0.5)
10	Dense	1024	-	relu	Dropout(0.5)
11	Dense	100	-	softmax	-

Table A.10: Structure of model 5 cl half size 4196 fc

layer	name	kernels	size of kernel	activation function	regularization
1	Conv2D	48	(11, 11)	relu	-
2	MaxPooling2D	-	(3, 3)	-	-
3	Conv2D	64	(5, 5)	relu	-
4	MaxPooling2D	-	(3, 3)	-	-
5	Conv2D	95	(3, 3)	relu	-
6	Conv2D	95	(3, 3)	relu	-
7	Conv2D	64	(3, 3)	relu	-
8	MaxPooling2D	-	(3, 3)	-	-
layer	name	neurons	-	activation function	regularization
9	Dense	512	-	relu	Dropout(0.5)
10	Dense	512	-	relu	Dropout(0.5)
11	Dense	100	-	softmax	-

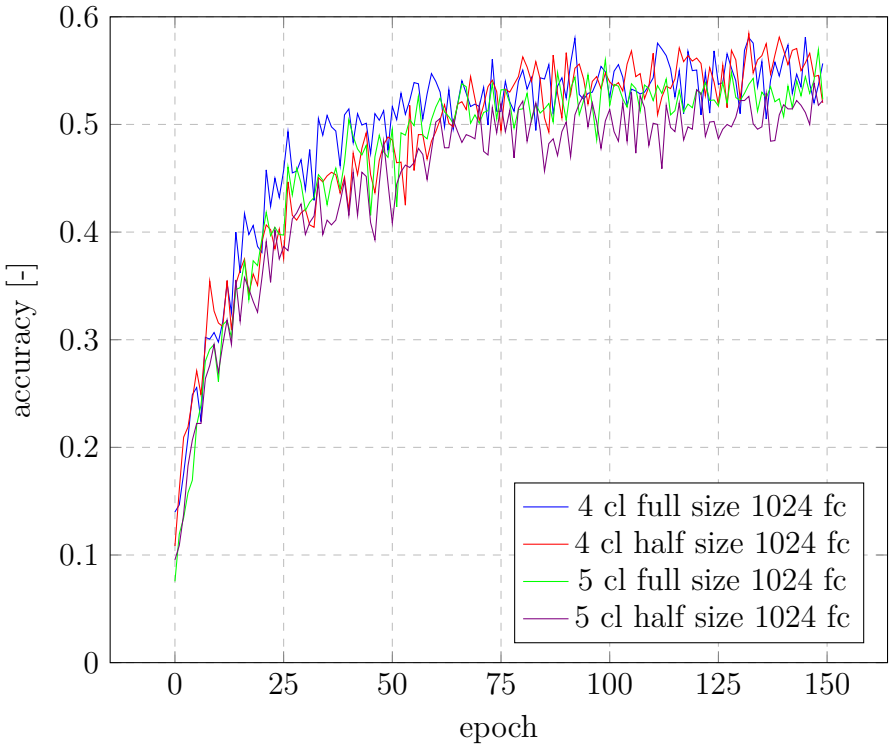

Figure A.1: Training accuracy of models with 1124 neurons in fully connected layer.

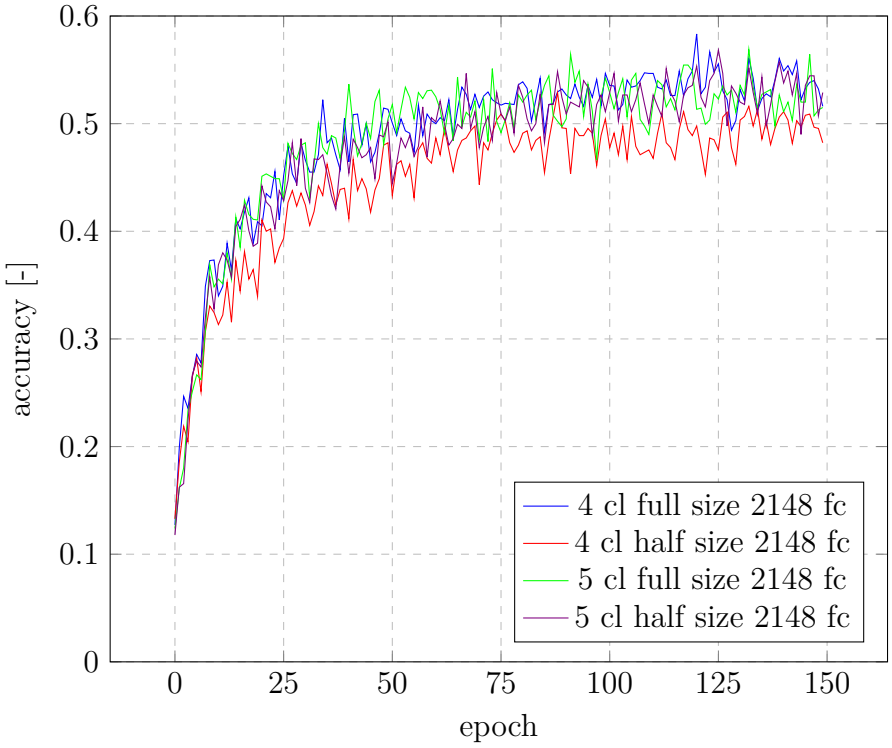

Figure A.2: Training accuracy of models with 2148 neurons in fully connected layer.

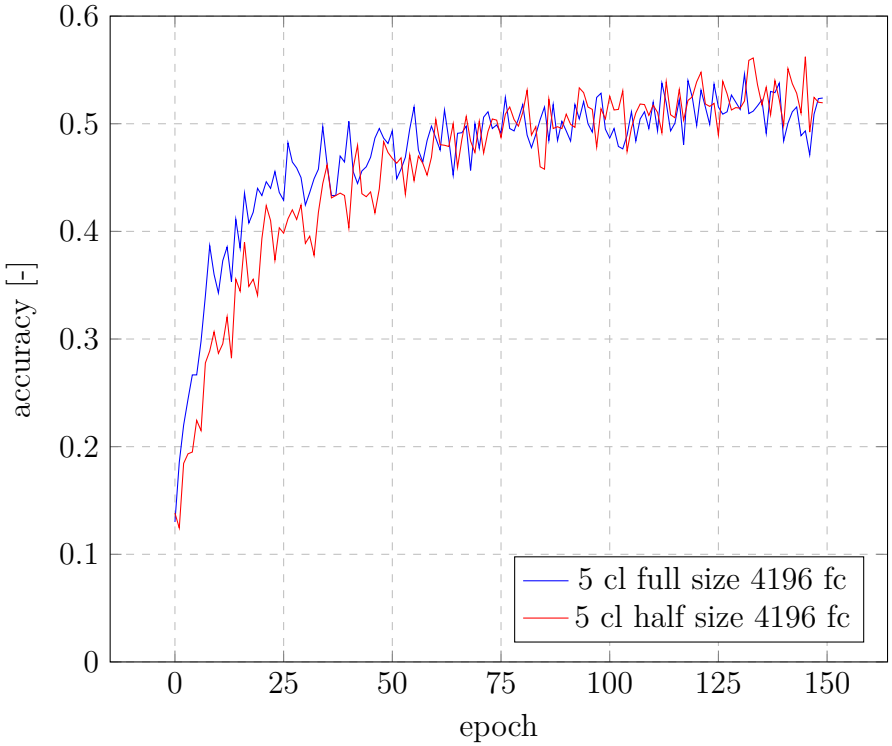

Figure A.3: Training accuracy of models with 4196 neurons in fully connected layer.

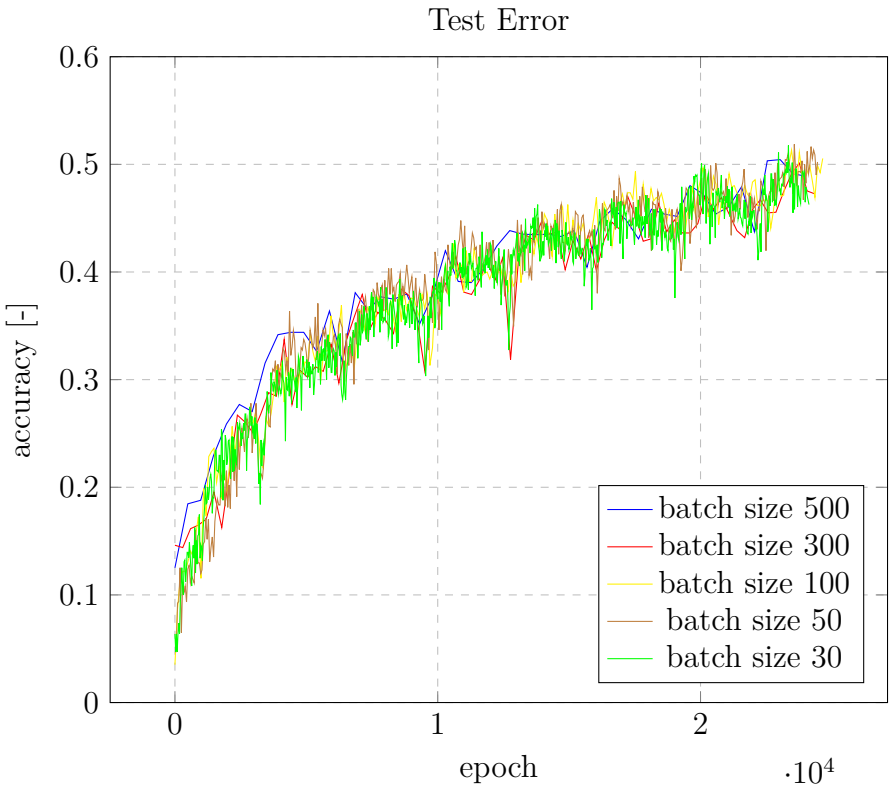

Figure A.4: Comaprison of accuracy for different size of training batch