Bachelor Project

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Deep Learning for Pattern Recognition of Brain Image Data

Nikita Tishin

Supervisor: MSc. Sebastián Basterrech Ph.D.
May 2018

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Tishin Nikita** Personal ID number: **452982**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Branch of study: **Computer and Information Science**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Deep Learning for Pattern Recognition of Brain Image Data

Bachelor's thesis title in Czech:

Aplikace hlubokého učení v rozpoznávání obrazových dat mozku

Guidelines:

Deep Learning paradigm has gained relevance last years in the community of Artificial Intelligence and Machine Learning. Several Deep Learning techniques have been successfully applied for computer vision, natural language processing, image processing, time-series analysis. In this project the student will analyze the performance of Convolutional Neural Networks for recognising patterns on brain imagery. In particular the student will apply Tensor Flow for developping the computational models and evaluate their performance in well-known benchmark problems and real datasets.
The main goal of the project is to develop a tool based in Deep Learning and CNN for recognising mental disor-ders according to brain images.
During the project the student should do the following steps:
1. Theoretical revision of Deep Learning techniques and current state in the field of Brain Imagery.
2. Theoretical study of Convolutional Neural Networks.
3. Implementation of Convolutional Neural Networks.
4. Evalution of CNN on at least two well-known benchmark problems. Sensitive analysis of the model parameters.
5. Application of CNN on real brain images, which will be provided by the superviser. The images can be MRI and/or fMRI.
6. Analysis the experimental results, including comparisons with the state of art and sensitive analysis of the CNN model.

Bibliography / sources:

[1] Bishop, Christopher M. Pattern recognition and machine learning. Springer, 2006.
[2] Juergen Schmidhuber, Deep Learning in Neural Networks: An Overview, Neural Networks, Vol. 61, pages 85-117, 2015. Doi: 10.1016/j.neunet.2014.09.003, available at: https://arxiv.org/abs/1404.7828
[3] Yoshua Bengio, Deep Learning of Representations: Looking Forward, Department of Computer Science and Operations Research, University of Montreal, Canada, 2013. Available at: http://goo.gl/OK0WV9 .
[4] Hastie, T., Tibshirani, R., Friedman, J., The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Second Edition, Springer, February 2009.
[5] Deep Learning information: http://deep learning.net.
[6] Shen H, Wang L, Liu Y, Hu D. Discriminative analysis of resting-state functional connectivity patterns of schizophrenia using low dimen- sional embedding of fMRI. Neuroimage. 2010;49:3110-21.
[7] Anderson JS, Nielsen JA, Froehlich AL, et al. Functional connec- tivity magnetic resonance imaging classification of autism. Brain. 2011;134:3742-54.
[8] Smith K. Brain imaging: fMRI 2.0. Nature. 2012; 484:24-6.

Name and workplace of bachelor's thesis supervisor:

Sebastian Basterrech, MSc., Ph.D., Artificial Intelligence Center, FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **12.01.2018** Deadline for bachelor thesis submission: **25.05.2018**

Assignment valid until: **30.09.2019**

Sebastian Basterrech, MSc., Ph.D.	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Ing. Pavel Ripka, CSc.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

I would like to express my gratitude to my supervisor Sebastián Basterrech, for the great feedback and encouragement. I am also very grateful to my family and friends for their support.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, 25.5.2018 .....

Abstract

In recent years Convolutional Neural Networks (CNNs) have became a most popular choice for image recognition problems. They provide a framework for representation learning, which allows for raw input processing without any manual feature engineering. The first goal of this thesis is a theoretical study and comprehension of CNNs. The second goal is evaluation and comparison of several CNN applied to the image classification. Experimental part includes two benchmark problems, i.e. MNIST database and CIFAR-10, as well as medical dataset consisting of recurrence plots created from the Electroencephalography (EEG) signals. In particular, CNNs are used to classify EEG data received from mentally healthy persons and persons having a mental illness.

Keywords: DL, Deep Learning, CNN, Convolutional Neural Networks, Supervised Learning, Electroencephalography, EEG, Recurrence Plot

Supervisor: MSc. Sebastián Basterrech Ph.D.

Abstrakt

V posledních letech konvoluční neuronové sítě (CNN) se staly nejpopulárnějším výběrem pro problémy rozpoznávaní obrazů. CNN poskytuji framework pro reprezentační učení, které umožňuje zpracování vstupních dat v původním tvaru bez ručního vytvoření příznaků. Za prvé, cílem teto práce je teoretické studium a pochopení CNN. Za druhé, cílem je vyhodnocení a porovnání několika CNN použitých na klasifikací obrazů. Experimentální část obsahuje aplikaci CNN na problémech MNIST a CIFAR-10, a také dataset z oblasti mediciny stvořený z recurrence plots EEG signálů. Konkretně CNN jsou použité na klasifikaci EEG dat získaných od duševně zdravých lidi a od lidi s duševními choroby.

Klíčová slova: DL, Hluboké účení, CNN, konvoluční neuronové sítě, Učení s učitelem, Elektroencefalografie, EEG, Rekurentní Graf

Překlad názvu: Aplikace hlubokého učení v rozpoznávání obrazových dat mozku

Contents

Figures

Tables

Chapter 1

Introduction

In the last decades the field of Machine Learning (ML) has been undergoing dramatic changes. It has witnessed the rise of Deep Learning (DL) and Artificial Neural Network (ANN) in particular. Some concepts that can be attributed to DL have been known since the end of XX century, but for various reasons they did not get much appreciation by the ML community. It has changed with the adoption of Convolutional Neural Networks (CNNs) in image processing problems. CNN is a learning system that consists of multiple computational layers that employ convolution operation. Consecutive applications of convolution transform the input of the system into its most informative representation in the context of problem being solved. Development of CNNs and their efficient Graphics Processing Unit (GPU) implementations [37] made some problems that were considered impractical to solve with ANNs computationally feasible [47]. At the same time, series of image recognition contests in 2011-2012 years were won by teams that were using CNNs [6]. This brought a lot of attention to the area of DL. One of the most known results from this period is the CNN that won large-scale object recognition ImageNet competition, outperforming other methods by a significant margin [29].

CNNs, however, do not work solely with images of the real world objects. They are naturally applied to problems involving any kind of grid-structured data. Medicine in particular works with a wide variety of data suitable for processing by CNNs, e.g. ultrasound, tomography or functional Magnetic Resonance Imaging (fMRI). There are also techniques for analyzing one-dimensional data, such as EEG, by increasing the original dimensionality in order to make them more suitable for CNN processing. In particular, Recurrence Plots [10] are used for analyzing dynamical systems. In this thesis Recurrence Plot technique is used to transform univariate EEG signal into two dimensional image data. As an instance of CNNs application in the medical field, they have been successfully applied to cancer detection on a large scale [7] [12].

The goal of this thesis is to analyze learning capabilities of CNNs and their performance in image classification. We evaluate CNN architectures on benchmark datasets MNIST [54] and CIFAR-10 [28]. In addition, a CNN model is applied on a classification problem in a real medical data. The

problem is to classify rest and task states based on the Recurrence Plot of the EEG signal.

The thesis is structured in a following way: introduction is followed by a second chapter providing a general overview of ML concepts that are necessary for understanding DL techniques. Third chapter presents ANNs – major area of DL and a foundation for the next chapter on CNN. Theoretical overview is followed by the implementation methodology description and a brief introduction EEG data.

Chapter 2

Basic Concepts of Machine Learning

This chapter introduces main concepts of Machine Learning – a field of computer science that employs statistical methods for making computer systems able to learn from data [25]. It covers topics that are essential to understanding complex learning systems like neural networks. Starting from the brief overview of most important machine learning concepts, this chapter concludes with model training and evaluation basics.

2.1 Machine Learning Systems

The area of **ML** is creating a system that can perform a specific task without being explicitly programmed [41]. Instead, it should be sufficient to provide such system with empirical data of the process which behavior is being studied. This can be stated formally [50]:

> "A computer program is said to learn from experience **E** with respect to some class of tasks **T** and performance measure **P**, if its performance at tasks in **T**, as measured by **P**, improves with experience **E**."

This definition captures all important aspects of a ML problem.

- **T, the task** – the problem a ML system is intended to solve. The most common tasks *regression* and *classification*. Regression models are trained to estimate unknown continuous variable based on given inputs. In classification a ML model learns to specify which of k categories some object belongs to.
- **P, the performance measure** is a quantitative measure of how well the model performs. Optimizing the performance measure is one of the main goals of machine learning. Usually the choice of the metric depends on the type of the task that machine learning system is dealing with. A a common practice is to use several metrics that give different perspectives on the model performance.
- **E, the experience**. In most ML algorithms experience is represented by a **dataset** – a collection of **data points**, that are represented by

real-valued vectors $\mathbf{x} \in \mathbb{R}^m$. Generally, ML problems require collecting empirical evidence and transforming it into the values that can be accepted by the learning algorithm. This process is commonly referred to as *feature engineering*. Although some branches of machine learning, like deep learning, deviate from this approach and make use of as much information as possible. For instance, a gray-scale image could be represented by a feature equal to the sum of its left-part pixel values subtracted from the sum of its right-part pixel values. Deep learning system would use the whole image as an input.

When it is possible, it is convenient to represent the dataset with a matrix. A dataset consisting of n m-dimensional samples forms a matrix $\mathbf{X} \in \mathbb{R}^{n \times m}$. Then i-th row of the $\mathbf{X}$ denotes a data point $\mathbf{x}_i$.

Learning algorithm is an algorithm that is able to produce ML models using input data. is Learning algorithms can be classified as **supervised** and **unsupervised** [25], albeit sometimes this boundary is too vague. Apart from that, there is also **reinforcement** learning, in which learning algorithms interact with dynamical systems "in real time" and receive feedback to evaluate the successfulness of their actions [48].

In supervised learning, each data point is associated with a **target value**, which provides information for learning and which the model tries to estimate when given a new, unseen input. There are no target values provided in unsupervised learning problems. Presence of the target value impacts learning methodology to a great extent. This thesis concentrates on supervised learning algorithms.

2.2 Supervised Learning

In supervised learning the task is to *estimate* (or to predict, which is the same in this context) one or more *outputs* (or *targets*) based on one or more *inputs* using empirical evidence (a *dataset*) that is collected beforehand [40].

Let $\mathbf{x} \in \mathbb{R}^p$ and $\mathbf{y} \in \mathbb{R}^q$. An input $\mathbf{x}$ causes a response in a form of the output $\mathbf{y}$. This relationship can be described in a general form:

$$\mathbf{y} = f(\mathbf{x}) + \epsilon. \tag{2.1}$$

ϵ is a random *error term*, and $f : \mathbb{R}^p \to \mathbb{R}^q$ is an arbitrary mapping from the **input space** to the **output space**. Assuming that there is a dependence of $\mathbf{x}$ on $\mathbf{y}$, it can me modeled with some function $\hat{f} : \mathbb{R}^n \to \mathbb{R}^m$ in a way that a chosen performance measure $\mathbf{J}$ is optimized with respect to the dataset $\mathbb{D} = \{(\mathbf{x}_i, \mathbf{y}_i) | \mathbf{x} \in \mathbb{R}^p, \mathbf{y} \in \mathbb{R}^q\}$.

2.3 Learning Algorithms

Learning is a process of improving the performance measure with respect to the given dataset and *generalizing* knowledge beyond that particular dataset [4].

2.3.1 Parametric Models

In case of supervised learning, training algorithms are provided with data that include the desired output of the system. Supervised learning algorithms are designed to make use of this information. Model is created or gradually improved through receiving explicit feedback on its performance in a form of quantitative performance measure.

Models can be classified as **parametric** and **non-parametric** based on the assumptions that should be made to apply them.

- Parametric models are based on the assumption that the unknown target function $f(\cdot)$ lies in **hypothesis space** – a set of parametrized functions $\{f(\mathbf{x};\theta)|\theta \in \mathbf{\Theta}\}$. Approximating function $\hat{f}(\cdot)$ is chosen from this set by finding a parameter configuration which optimizes the performance measure. *Linear regression* [5] and *ANNs* [15] are examples of parametrized models.

- Non-parametric learning algorithms make almost no assumptions about the unknown function of interest, and do not put constraints on the hypothesis space. Non-parametric models include *K-Nearest neighbors* [25] and *decision trees* [25].

While the family of non-parametric algorithms is diverse, parametric learning is usually stated and solved in terms of *mathematical optimization*. The problem of finding optimal θ values with respect to the cost function $J : \Theta \to \mathbf{R}$ and the dataset $\mathbb{D}$ can be formulated as

$$\min_{\theta\in\Theta} \quad J(\theta;\mathbb{D}). \tag{2.2}$$

The solution to this problem depends on the special properties J might or might not possess [44].

2.3.2 Cost functions

The choice of the cost function depends on the specific ML task and sometimes on the properties of the given dataset. Most common cost functions for *regression* and *classification* are the mean squared error and the cross entropy.

- **Mean squared error** is generally used in regression problems [1] and measures the squared euclidean distance between the vector consisting of target values and the vector of estimated values. It is defined as follows:

$$J(\theta;\mathbb{D}) = \frac{1}{|\mathbb{D}|} \sum_{(\mathbf{x}_i,\mathbf{y}_i)\in\mathbb{D}} \|\hat{f}(\mathbf{x};\theta) - \mathbf{y}\|_2^2. \tag{2.3}$$

- **Cross entropy** [42] is popular in classification tasks, where the output of the model simulates the simple or multivariate Bernoulli distribution [8]. That is, the model outputs probabilities of the input belonging

to some classes. Loosely speaking, it measures the difference between two probability distributions. It is defined as follows:

$$J(\theta; \mathbb{D}) = -\sum_{(\mathbf{x}_i, \mathbf{y}_i) \in \mathbb{D}} \sum_{j=1}^{p} \mathbf{y}_{i,j} \log \hat{f}(\mathbf{x}_i; \theta)_j. \tag{2.4}$$

2.3.3 Generalization

Aside from just improving performance measure it is desirable to achieve **generalization** [4]. Model is able to generalize if it performs well when given previously unseen inputs. To measure generalization error, dataset is divided into *training* and *test* parts. Data for testing do not participate in the learning process and are intended solely for model evaluation. A supervised learning model is **underfitting** if its error on the training dataset is undesirably high [15]. A model is **overfitting** if it achieves low training error but the test error is far higher than the training error [15]. Models with high number of parameters are able to overfit the training dataset by "memorizing" it. On the other hand, models with fewer number of parameters can be insufficient and prone to underfitting.

Test set error is only an approximation of the true error, which denotes the error on all previously unknown inputs. Sufficiently big test set can provide an adequate approximation, but if the test set does not contain enough samples this estimate is not reliable. **Cross-validation** [3] is a family of algorithms addressing this issue. The dataset is randomly divided into train and test (validation) parts, model is trained and evaluated. This process is repeated several times and the final test error is approximated with the average validation error. In the area of DL, cross-validation might be unfeasible to perform, because repeated training of a DL model can take unacceptable amount of time.

2.3.4 Regularization

Regularization is a way of constraining the hypothesis space of a model for the purpose of reducing the generalization error without increasing the training error.

In supervised parametric learning tasks regularization takes form of **regularization penalty** function $\Omega : \Theta \to \mathbb{R}$:

$$\min_{\theta \in \Theta} \quad L(\theta; \mathbb{D}) = J(\theta; \mathbb{D}) + \lambda \Omega(\theta), \tag{2.5}$$

where $L\Theta \to \mathbb{R}$ is the **total cost function** and λ is a *hyperparameter* controlling the regularization strength. Hyperparameters are variables which does not participate in the learning process and should be predefined when constructing the model. Higher values of λ give the regularization part of the loss function bigger weight.

Common penalties are

- L_2 loss:
$$\Omega(\theta) = \|\theta\|_2^2 = \theta^T\theta, \tag{2.6}$$
- L_1 loss:
$$\Omega(\theta) = \|\theta\|^1 = \sum_i |\theta_i|. \tag{2.7}$$

These penalties force optimization process to prefer solutions in which some parameter values are low or zero, thereby dynamically reducing the number of active parameters.

2.4 Evaluation of a Binary Classifier

sometimes be unintuitive and hard to comprehend. However, there are specialized interpretable metrics in the classification setting, designed to show classifier's performance from different perspectives.

2.5 Confusion Matrix

Binary classifier is a supervised learning model which maps input features to one of the two class labels, which are commonly called **positive** and **negative** label. Often a binary classifier outputs values from the $[0, 1]$ interval. This value can be interpreted as a probability of the input falling into the positive class. Specific evaluation system has been developed for binary classification. It is based on the **confusion matrix**, which displays all possible outcomes of the label estimation:

	True label positive	True label negative
Predicted label positive	# of true positives, TP	# of false positives, FP
Predicted label negative	# of false negatives, FN	# of true negatives, TN

Table 2.1: Confusion matrix for binary classification

- **Accuracy** indicates how many samples were classified correctly:
$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.8}$$
- **Precision**, or positive predictive value, measures the correctness of predicting positive condition:
$$PPV = \frac{TP}{TP + FP} \tag{2.9}$$
- **Sensitivity**, also called **recall**, or true positive rate, measures the proportion of correctly classified true samples:
$$TPR = \frac{TP}{TP + FN} \tag{2.10}$$

- **False alarm**, or false positive rate, measures the proportion of incorrectly classified positive samples to all negative samples:

$$FPR = \frac{FP}{TN + FN} \tag{2.11}$$

2.5.1 Receiver Operating Characteristic Curve

ROC curve is created by computing true positive rate and false positive rate for different thresholds applied on the output of the binary classifier. ROC curve can be represented with a set of pairs:

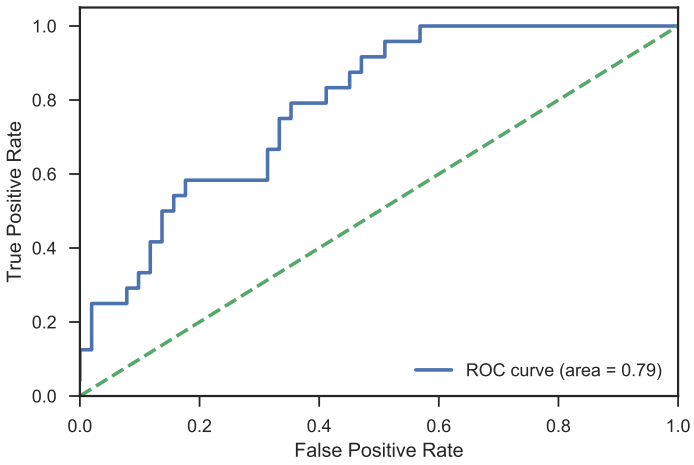


Figure 2.1: An example of the ROC curve.

2.5.2 Precision Recall Curve

PR curve is very similar to the ROC curve. PR curve uses precision (positive predictive value) instead of false alarm, and differs in the axis relative positions.

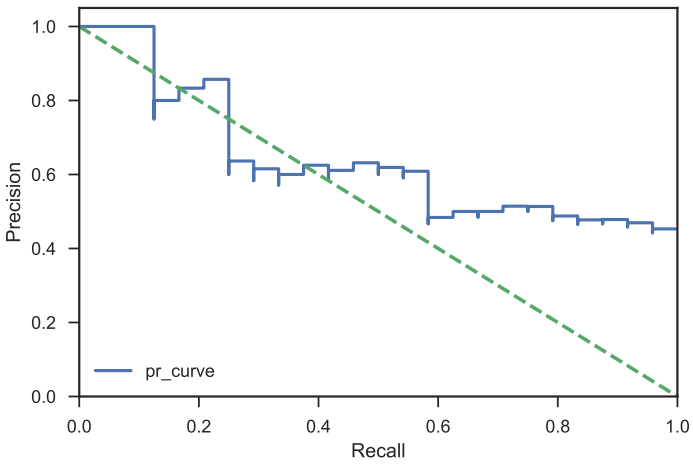


Figure 2.2: An example of the PR curve.

Chapter 3

Artificial Neural Networks

A field of machine learning called **DL** is based on the idea of capturing hierarchical structure of data with composite models [32]:

> "Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction."

ANN is a parallel distributed connectionist system made up of simple processing units (commonly referred to as **neurons**) that has a natural ability for storing experimental knowledge and making it available for use [19].

ANN is a model consisting of multiple computational layers stacked on top of each other. Successive layers are thought to learn features of increasing abstraction, looking "deep" into the data. At the origins of deep learning, biological neural networks were a big source of inspiration for artificial neural network research. ANNs are a major area of deep learning study and are successfully applied in many areas, such as image recognition [11], speech recognition [17], natural language processing [14] and bioinformatics [55].

Two types of models can be distinguished by their topology: **feedforward** and **recurrent** neural networks. In feedforward networks information flows from the input to the output through computational layers consisting of processing units and there is no feedback connections. If they are present and intermediate outputs are redirected back to previous layers or stored for later use, network is called recurrent. Thus, a feedforward neural network can be represented with *directed acyclic graphs*, while recurrent ones are represented with *directed cyclic graphs*. This text concerns only feedforward networks.

This chapter covers notions which constitute the basis of ANNs, followed by multilayer perceptrones – the foundation stone of deep learning. The rest of the chapter describes the process of learning with neural networks.

3.1 Basic Concepts

ANNs originated as systems inspired by biological neural networks. They were intended to simulate animal's brain. So, by analogy, they consist of neurons and connections between them.

Artificial neuron was designed as a mathematical model of the biological neuron. It is a function transforming n real input "stimuli" to the single "response value" $f : \mathbb{R}^n \to \mathbb{R}$. The most common definition of the neuron is

$$f(\mathbf{x}) = \phi(\mathbf{w}^T\mathbf{x} + b), \tag{3.1}$$

where $\mathbf{w} \in \mathbb{R}^n, b \in \mathbb{R}$ are the neuron's **weights** and **biases** respectively, and $\phi : \mathbb{R} \to \mathbb{R}$ is the activation function, which is described below. Weights and biases are the neuron's parameters, and their values are determined in the process of learning. Bias makes this transformation *affine*, allowing to learn a bigger variation of functions.

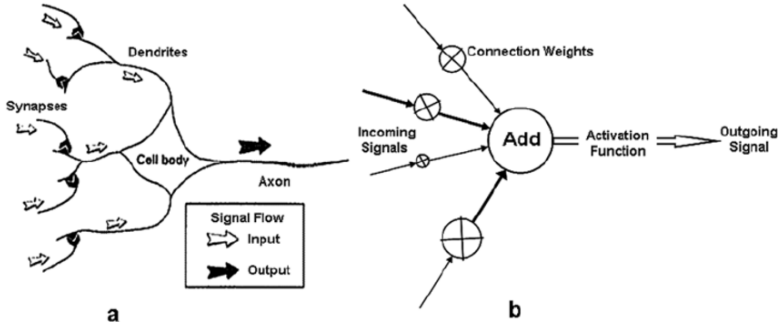
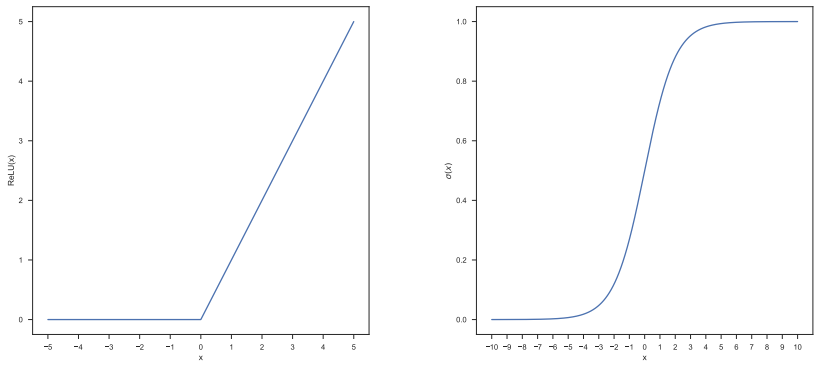


Figure 3.1: A biological neuron (a) compared to an artificial neuron (b). The analogy is clear: both neurons receive multiple signals and aggregate them to a single output that is transferred further on. Source: [18].

Activation function transforms the output of a neuron. It is applied element-wise on the output vector. The purpose of an activation function is to introduce non-linearity to the model in order to learn complex non-linear functions. Without activation functions hypothesis space of a neural network would be constituted only from linear functions. Popular examples of activation function include **ReLU** (Rectified Linear Unit) $\text{ReLU}(x) = max(0, x)$ and **sigmoid** $\sigma(x) = \frac{e^x}{e^x+1}$.



(a) : ReLU activation **(b) :** Sigmoid activation

Figure 3.2: Popular activation functions

Layer is a collection of neurons which are grouped to process multi-dimensional inputs to multi-dimensional outputs. ANNs are usually described in terms of layers, since it is more convenient due to the bigger scale and abstraction compared to artificial neurons. A notable example of a layer is a **fully-connected layer**. In a fully-connected layer neurons receive input from *each* neuron of the previous layer. This allows for convenient simultaneous computation of all neuron outputs of the layer via matrix multiplication:

$$f(\mathbf{x}) = \phi(\mathbf{W}\mathbf{x} + \mathbf{b}), \tag{3.2}$$

where $\mathbf{W}$ and $\mathbf{b}$ are individual neuron's weights and biases collected into a matrix and a vector respectively.

Network topology is a configuration of multiple chained connected layers is called . Two special layers are always present in the network: **input** and **output** ones. An input layer holds values of the input vector and is connected to the next layer only to pass it forward. An output layer is the last layer in the chain and its outputs are the final result of the input processing done by the network. Layers between them are called **hidden**, because their outputs are only intermediate and are not used outside of the model. The number of hidden layers is called the network's **depth**. The number of neurons in the layer is referred to as its **width**.

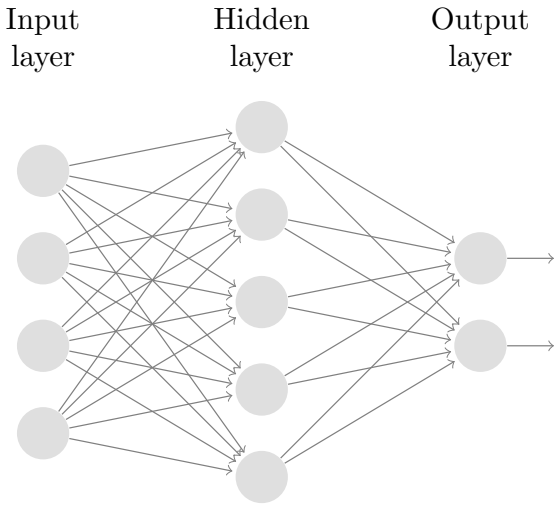


Figure 3.3: A simple artificial neural network example. It has only one hidden layer consisting of 5 units, so the depth of the model is one. The hidden layer is fully-connected.

It is worth noting that modern deep learning research and applications deviated from the initial biological and neuroscientific origins in order to develop more task-specific systems that concentrate on achieving better performance using various heuristics. Modern Deep Learning primarily relies on mathematics and computer science [15].

3.2 Feedforward Neural Networks

3.2.1 Single-Layer Perceptron

Single-layer perceptron [39] is the simplest kind of a feedforward neural network. In the basic form, it consists of only one artificial neuron. It can be extended to produce multi-valued output via additional units for each output element, but it still would not have any hidden layers. This type of neural network has very limited learning potential. Due to the absence of hidden layers, outputs are just a linear combination of inputs, so the single-layer perceptron will only be able to approximate linear relationships [35].

Perceptron function is the same as for the artificial neuron in the equation (3.1). Perceptrons can be trained by a simple learning algorithm that is usually called the delta rule [52]. It calculates the errors between calculated output and sample output data, and uses this to create an adjustment to the weights.

Algorithm 3.1 Delta rule learning

Require: $f : \mathbb{R}^n \to \mathbb{R}$ ▷ Perceptron function
Require: α ▷ Learning rate parameter, discussed later
Require: $\mathbb{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ ▷ Training dataset, $y_i \in \mathbb{R}$ is the desired response to the input vector $x_i \in \mathbb{R}^n$
for $(\mathbf{x}_i, y_i)$ in $\mathbb{D}$ **do**
$\hat{y}_i \leftarrow f(\mathbf{x})$
$\mathbf{w} \leftarrow \mathbf{w} + \alpha(y_i - \hat{y}_i)\phi'(\mathbf{wx})x_i$
end for

3.2.2 Multilayer Perceptron

Multilayer perceptron (MLP) is a neural network that has at least one hidden layer and all layers are fully-connected. MLP is the basic tool and the basis of many advanced techniques in the DL. It is a cornerstone of deep learning.

Mathematically, MLP is a composite function that alternates affine and non-linear transformations, mirroring the graphical structure of the network []:

$$f(\mathbf{x}) = g(\mathbf{W}_l\phi_{l-1}(\mathbf{W}_{l-1}(\ldots\phi_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)\ldots) + \mathbf{b}_{l-1}) + \mathbf{b}_l), \tag{3.3}$$

where $\mathbf{W}_i, \mathbf{b}_i, \phi_i$ are the i-th layer's weights, biases, and activation function respectively, and $g(\cdot)$ is the output function. Figure 3.3 depicts a multilayer perception with a single hidden layer.

MLP can be used in a broad range of tasks – according to the *universal approximation theorem* [21], a MLP with at least one hidden layer and non-linearity containing a finite number of units can approximate any continuous function with any desired non-zero error given appropriate parameters. However, the theorem is not constructive and it does not provide a way to find

such network configuration and it's parameters. In practice optimization algorithm is not guaranteed to find parameters corresponding to the desired function. It means that designing and training a network may not lead to desired approximation quality. Nevertheless it is an important result that shows ANN's rich potential.

In practice MLP has a few problems that balance it's attractiveness as a universal approximator. One of the biggest downsides of this model is high number of trainable parameters due to the fact that all layers are fully connected. Number of parameters can become unfeasible to train for high dimensional data or wide hidden layers. It also limits the depth of the model. Not only deep and wide perceptrones are computationally hard to train, but they tend to overfit if dataset is not large enough because of their high capacity.

3.3 ANN Learning

ANN is a parametric model. In order to get meaningful results for the task we should find appropriate values for its parameters – weights and biases. Just like in any other ML problem, ANN requires choosing a performance measure and an algorithm which will allow the network to improve this measure with respect to the dataset. It can be achieved by means of mathematical optimization.

In this section we introduce two algorithms essential for ANN learning: gradient descent, which adjusts network parameters in a way that improves performance measure, and back-propagation, which computes the gradient of the cost function necessary for gradient descent.

3.3.1 Gradient Descent Methods

The optimization problem of training a neural network can be stated formally:

$$\text{minimize} \quad L(\theta; \mathbb{D}) \quad \text{w. r. t.} \quad \theta \in \Theta \tag{3.4}$$

where $\mathbb{D}$ denotes the set of training samples, θ denote all parameters of the model under the parameter space Θ and L is the real loss function. Evaluating L naturally involves computing model output on each training sample $\mathbf{x} \in \mathbb{D}$. In the case of ANNs it makes performance function non-linear, which leads to complex and often non-convex optimization problems. Such problems can be solved with iterative gradient-based algorithms.

Gradient of a real continuous function $f : \mathbb{R}^n \to \mathbb{R}$ is the column vector of its partial derivatives:

$$\nabla f(\mathbf{x}) = \left[\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right]^T. \tag{3.5}$$

Gradient is a special case of the **Jacobian matrix J**, which is defined for a

multi-dimensional real continuous function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}. \tag{3.6}$$

- **Gradient Descent (GD)** is an iterative procedure for greedy optimization of arbitrary functions. Informally, gradient is the vector that points in the direction of the greatest rate of increase of the function. If it is desirable to achieve maximum possible decrease in a value of a function at some point $\mathbf{x}$, the gradient of the function should be subtracted from this point. This operation is referred to as **update rule**. Update rule is repeatedly applied for a certain number of steps or until some stopping condition (e.g. relative difference of the previous and successive function values is under a certain threshold) is satisfied. This basic approach from optimization theory applies to ANN training. Training process involves computing gradient many times, so this step should be as fast as possible. In neural networks the back-propagation algorithm is used for gradient computation. It is the subject of the next section.

  GD update rule:

  $$\theta_{k+1} = \theta_k - \alpha \nabla_\theta L(\theta; \mathbb{D}), \tag{3.7}$$

  where $\alpha \in \mathbb{R}$ is the **learning rate**, a hyperparameter used to control the magnitude of the update, which influences convergence of the algorithm.

  In the case of ANNs GD encounters problems which make the optimization particularly difficult: high number of variables (weights and biases combined) and typically high cardinality of a training set, which makes computing step of the gradient descent computationally infeasible, since it requires "feeding" the whole training dataset to the network.

- **Batch Stochastic Gradient Descent (SGD)** performs gradient update on the random subset of the original dataset, called batch. It addresses the GD issues. Main idea behind SGD is that computing the gradient on random batches gives sufficient approximations of the real gradient. This approach allows much faster computation in exchange for precision in calculations. Because of this the loss function usually fluctuates and does not decrease monotonically. SGD favors exploration over greedy exploitation of the loss function surface because of slightly random suboptimal gradient directions. In the context of SGD one pass through all batches is called an **epoch**, and processing a single batch is referred to as a **step**.

  Step of the mini-batch SGD:

  $$\theta_{k+1} = \theta_k - \alpha \nabla_\theta L(\theta; \mathbb{D}^{(\mathbf{I}_k)}), \tag{3.8}$$

  where $\mathbf{I}$ denotes a vector of random dataset indexes. Length of this vector is determined by the **batch size** hyperparameter.

- SGD **with momentum** [38] is intended to simulate the movement of the ball that has mass. It accumulates momentum as the parameter point displaces downhill, accelerating in the direction of a minimum, albeit it does not stop there, oscillating around a minimum when it is reached. It is implemented by subtracting the scaled update vector of the previous step u_k:

$$u_k = \gamma u_{k-1} + \alpha \nabla_\theta L(\theta; \mathbb{D}^{(\mathbf{I}_k)}), \tag{3.9}$$
$$\theta_{k+1} = \theta_k - \gamma u_k, \tag{3.10}$$

  where $\gamma \in \mathbb{R}$ hyperparameter controls the power of the momentum.

- **Adaptive gradient (Adagrad)** [26]adapts the learning rate of each parameter individually, performing bigger updates for the parameters that change little in time, and vice versa. It's update rule is defined for each i-th parameter:

$$\theta_{k+1}^{(i)} = \theta_k^{(i)} - \frac{\alpha}{\sqrt{G_k^{(i,i)} + \epsilon}} \nabla_\theta L(\theta^{(i)}; \mathbb{D}^{(\mathbf{I}_k)}), \tag{3.11}$$

  where $G_k \in \mathbb{R}^{p \times p}$ is a diagonal matrix with its diagonal entries equal to the sum of the squares of the $\theta(i)$ gradients up to the step k and ϵ is a small number that helps to avoid division by zero.

- **Adaptive moment estimation (Adam)** [27] is an that algorithm makes use of both adaptive learning rate and momentum techniques. Adam computes decaying averages of past gradients m_k and past squared gradients v_k:

$$m_k = \beta_1 m_{k-1} - (1 - \beta_1) g_k, \tag{3.12}$$
$$v_k = \beta_2 v_{k-1} - (1 - \beta_2) g_k^2, \tag{3.13}$$

  which are then corrected to avoid biasing towards zero:

$$\hat{m}_k = \frac{m_k}{1 - \beta_1^k}, \tag{3.14}$$
$$\hat{v}_k = \frac{v_k}{1 - \beta_2^k}, \tag{3.15}$$

  where β_1 and β_2 are decaying rate hyperparameters. The final update rule for Adam looks like this:

$$\theta_{k+1} = \theta_k - \frac{\alpha}{\sqrt{\hat{v}_k} + \epsilon} \hat{m}_k. \tag{3.16}$$

3.3.2 Back-propagation Algorithm

Any kind of gradient descent algorithm requires computing the gradient numerous times. Evaluating the gradient of composite functions can be computationally expensive, and a loss function can potentially be quite

complex as it involves evaluating whole neural network on many data samples. **Back-propagation (BP)** algorithm provides an efficient way of evaluating the gradient [33].

BP utilizes chain rule in order to compute loss function. The algorithm can be considered as a specialized version of *reverse accumulation automatic differentiation* [13]. Reverse accumulation traverses the expression of chain rule from outside to the inside, computing all subexpressions along the way.

Reverse accumulation computes recursive relation:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial w_n}\frac{\partial w_n}{\partial w_{n-1}}\cdots\frac{\partial w_1}{\partial x}, \tag{3.17}$$

$$\frac{\partial f}{\partial w_i} = \frac{\partial f}{\partial w_{i+1}}\frac{\partial w_{i+1}}{\partial w_i}. \tag{3.18}$$

Neural network can be represented as a **computational graph** of th Back-propagation is designed in such way that it visits each node of the network once, avoiding repeated computations that arise while evaluating the gradient of compound function. The amount of computation required for BP scales linearly with the size of the computational graph the network.

Back-propagation always stars with a *forward pass*, which computes the error on one training sample. It has to keep all intermediate computation results as they are used to calculate the gradient.

Algorithm 3.2 Forward propagation algorithm for MLP [15]. Computer the loss for the input and target value pair.

Require: d, network depth
Require: $\mathbf{W}_i, i \in 1, \ldots, d$, individual layer weights
Require: $\mathbf{b}_i, i \in 1, \ldots, d$, individual layer biases
Require: $\mathbf{x}$, the input vector
Require: $\mathbf{y}$, the target vector
  $\mathbf{h} \leftarrow \mathbf{x}$
  for $l = 1, \ldots, d$ **do**
    $\mathbf{a}_l \leftarrow \mathbf{W}_l\mathbf{x} + \mathbf{b}_l)$
    $\mathbf{h}_l \leftarrow \phi(\mathbf{a}_l)$
  end for
  $\hat{\mathbf{y}} \leftarrow \mathbf{h}_d$

Once the error is evaluated, the *backward pass* begins. It is called so because it traverses from the output of the network to the inputs. From the point of symbolical computations, the expression of the chain rule is traversed from the outermost subexpression to the inner ones.

Algorithm 3.3 Backward propagation algorithm for MLP [15]. Computes the loss function's gradient with respect to the network parameters.

Require: d, network depth
Require: $L(\hat{\mathbf{y}}, \mathbf{y})$
  $\mathbf{g} \leftarrow \nabla_{\hat{\mathbf{y}}} L$
  for $l = d, d-1, \ldots, 1$ **do**
    $\mathbf{g} \leftarrow \mathbf{g} \odot \phi'(\mathbf{a}_l)$
    $\nabla_{\mathbf{b}_l} L = \mathbf{g} + \alpha \nabla_{\mathbf{b}_l} \Omega(\theta)$
    $\nabla_{\mathbf{W}_l} L = \mathbf{g}\mathbf{h}_{l-1}^T + \alpha \nabla_{\mathbf{W}_l} \Omega(\theta)$
    $\mathbf{g} \leftarrow \nabla_{\mathbf{h}_{l-1}} L = \mathbf{W}_l^T \mathbf{g}$
  end for

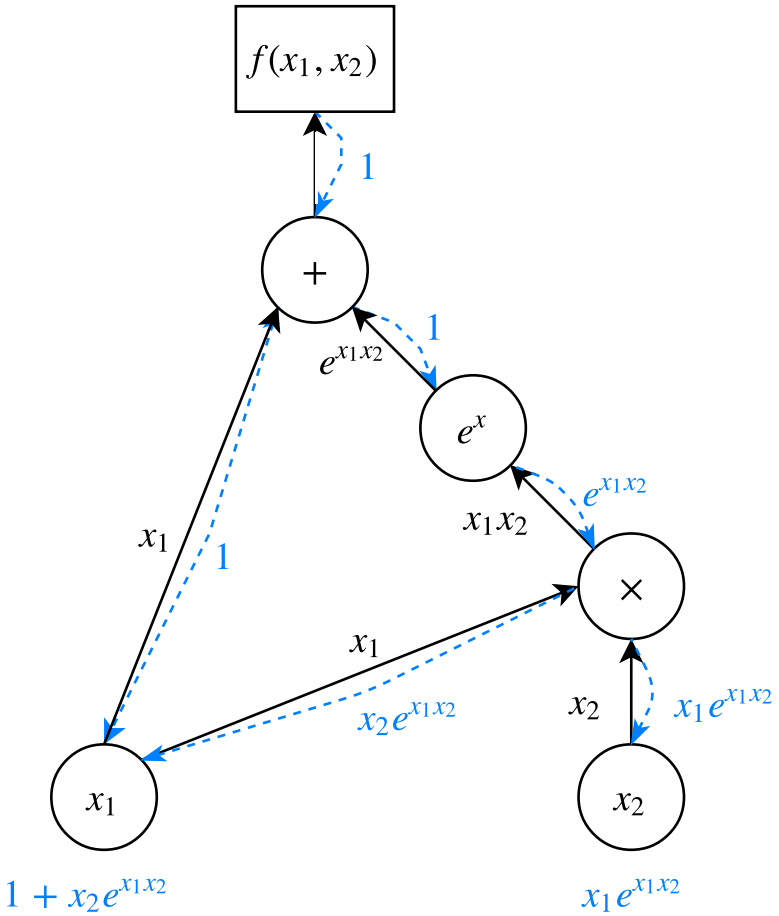


Figure 3.4: A computational graph of the function $f(x_1, x_2) = x + e^{x_1 x_2}$. Black and blue lines depict forward and backward pass respectively. Each computational node stores its forward and backward pass values, using them in a computation of local derivatives. The gradient is accumulated until input nodes are evaluated resulting in $\frac{\partial f}{\partial x_1} = 1 + x_2 e^{x_1 x_2}$ and $\frac{\partial f}{\partial x_1} = x_1 e^{x_1 x_2}$.

3.3.3 Regularization

In addition to L_1 and L_2 losses two most popular regularization techniques in the area of ANN are dropout and early stopping.

Dropout [46] is a regularization technique specific to neural networks. During the training, network units are randomly removed along with all incoming and out-coming connections. Dropout is controlled with the hyper-parameter $p \in [0, 1]$ which denotes the probability that a neuron will be kept in the network. After the training all nodes are inserted back into the network. Dropout reduces overfitting because dropped nodes have been learning only from the part of the data.

Early stopping [4] is a regularization method that can be used in any iterative optimization algorithm. Usually stochastic gradient descent runs for some predefined number of epochs. Once a *stopping criterion* holds true, the training process immoderately stops. Stopping criterion monitors training error and indicates if its value does not change with successive iterations, which means that the algorithm has stuck in a local minimum and further parameter adjusting is unnecessary and leads to overfitting.

Chapter 4

Convolutional Neural Networks

CNN is a special kind of ANN that uses **convolution** operation in at least one hidden layer. They are successfully used at different tasks involving processing data with grid-based structure, especially images.

This chapter begins with a description of convolution, which is a cornerstone to CNNs. It proceeds with technical details of convolutional layers and CNN training process and concludes with a discussion of main principles behind CNNs.

4.1 Convolution Operation

4.1.1 Convolution in Mathematics and Image Processing

Convolution is an operation of two functions producing a third function. In the most general case convolution is defined for continuous functions, but DL adopted its discrete version.

Let $f : \mathbb{Z} \to \mathbb{R}$ and $g : \mathbb{Z} \to \mathbb{R}$ be real valued functions and $D_f \subseteq \mathbb{Z}$ be the domain of $f(\cdot)$. Convolution is denoted with asterisk and is defined as follows:

$$(I * K)(t) = \sum_{a \in D_I} I(a)K(t - a). \tag{4.1}$$

In deep learning terminology I is called the **input**, and K is called the **kernel**. Values of the kernel are called weights as in traditional ANNs.

In ML practice CNNs work with images, which are typically represented as rectangular arrays of real values. In such case, convolution is two-dimensional and discrete. Convolution is also commutative, which means that the input can be shifted instead of the kernel. After these modifications we obtain the following definition:

$$(I * K)(i, j) = \sum_{1}^{m} \sum_{1}^{n} I(i - m, j - n)K(m, n). \tag{4.2}$$

$I \in \mathbb{R}^{w \times h}$ is the input image, $K \in \mathbb{R}^{m \times n}$ is the kernel and i, j are the output image pixel coordinates.

It is also worth mentioning that most modern deep learning libraries use **cross-correlation** instead of the convolution. The only difference between them is that cross-correlation does not "flip" the kernel:

$$(I * K)(i,j) = \sum_{1}^{m}\sum_{1}^{n} I(i+m, j+n)K(m,n). \tag{4.3}$$

This fact simplifies training process and does not influence learning results. If the kernel is flipped, then weights learned by CNN would simply have different indexes.

Convolution has many applications in different areas. Most importantly for the purposes of this text – in statistics, signal and image processing. In those areas it is often used for mixing two functions, filtering the original signal or applying various weighting functions to the input in order to highlight particular patterns or features of the input. CNN also use this operation for the purpose of feature detection. The important part is that networks not just use predefined hand-crafted kernels, but learn them in order to transform the input to the most informative *representation*. In deep learning terminology the output of convolution is sometimes denoted as the **feature map** [15].

4.1.2 Convolution in Image Processing

As it was previously mentioned, convolution is widely used in image processing. Using different kernels (also called *filters*), it is able to detect various image features, such as edges and blobs. Furthermore, it is used for changing image's visual representation, e.g. blurring or sharpening the picture.

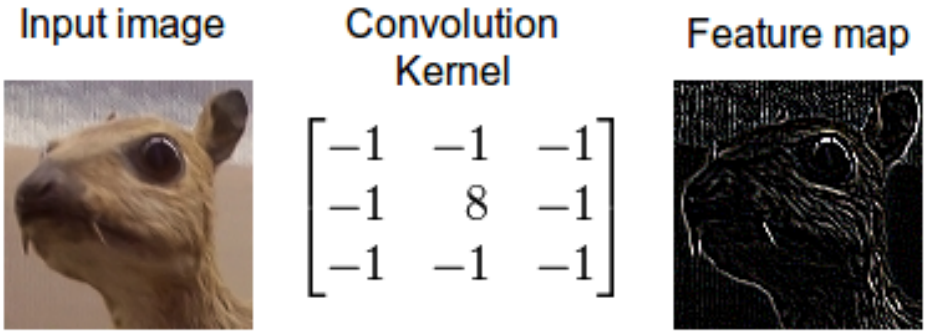


Figure 4.1: Edge detection with convolution [9]. Feature map (right) with highlighted edges is a result of convolution with kernel (middle) applied on the original image (left).

In this context, usage of convolution in neural networks is a heuristic adopted from the area of digital image processing. Taking advantage of the special form of the data, i.e. images, we can assume that specialized techniques will be more effective than universal multilayer perceptrones.

The main difference between standard image processing algorithms and CNNs is that in the latter case kernel's weights are learned through optimiza-

tion algorithm. This allows neural networks to learn specialized filters for each task.

4.1.3 Convolution in Neuroscience

Another source of inspiration for CNN design is early neuroscientific research on how mammalian visual system works, especially the study [23] of cat's brain responses to visual stimuli. DL adopted some concepts revolving around the part of the brain called **V1**, or **primary visual cortex**.

In a very simplified description of the process of visual input processing, image in the form of light stimulates the area of the eye called **retina** and eventually this signal reaches V1. V1 is known to have a two-dimensional structure in which each neuron responses only to certain region in retina – neuron's **local receptive field**. Despite that V1 is the most known part of this process, it is supposed that the input signal undergoes through several systems similar to V1 in a repeated fashion. Multiple layer stacking technique in DL mirrors those ideas about the visual system. After several layers of processing, the input reaches cells that individually respond to particular concepts disregarding the way they were seen, e.g. seeing someone on the photograph or in person. In CNN it is done through the output layer, which learns to take values depending on the last feature map and the target output.

4.2 Convolutional Network Architecture

4.2.1 Convolutional Layer

In CNN convolution is used to produce a feature map from the input, which can be the original image or an another feature map. The main goal of using convolution in ANNs is to take advantage of special structure of the input and to learn how to transform it to the most informative form.

In practice, convolutional layer's behavior is controlled with a set of hyperparameters, which brings flexibility to the design of neural networks and allows to adapt them to various problems:

- **Kernel size** defines the dimensions of the convolution kernel. It controls the area of the input to which neurons are sensitive. Choice of the appropriate value for this parameter almost always depends on the dataset. One way to do this is setting the kernel shape of the first layer according to the scale of images to capture important details, e.g. edges. However, no rule of thumb exists for deeper layers and the optimal kernel size is determined experimentally. In the case when the input contains multichannel images or any three-dimensional data kernel itself is often three-dimensional.

- **Number of kernels** controls the number of dimensions of the layer output, since each kernel would produce distinct feature map. Increasing number of kernels might help to reduce information loss in architectures

where feature map size tends to decrease with each layer. It also controls the capacity of the model – with the number of kernels the total number of trainable parameters grows.

- **Padding**: convolution is undefined on the borders of the input, since some part of the kernel can not be matched with any input value. Tn order to overcome this problem and apply convolution in corner cases, input can be framed with zeros. The number of the input values for which the convolution is defined directly influences the size of the output, so padding can be used to control it.
- **Stride** controls the "step" of the convolution filter. Stride value of two would mean that after applying convolution on some pixel, two pixels in all dimensions are skipped. By manipulating stride we can regulate overlapping of different receptive fields and reduction of the output size. Let n_{in}, n_{out}, k, p, s be the total number of the inputs and outputs, total kernel size, padding size and the stride, resp. Then the following relationship holds true [15]:

$$n_{out} = \lfloor \frac{n_{in} + 2p - k}{s} \rfloor + 1. \tag{4.4}$$

4.2.2 Pooling

A typical convolutional layer consists of three stages:

1. Applying convolutions to produce intermediate results.
2. Passing intermediate results through non-linear activation function, like in the standard multilayer perceptron. It is sometimes called *detection stage*.
3. Applying a **pooling function**.

Pooling function replaces rectangular areas of the input with their summary. It can be viewed as a non-linear downsampling method.

Let $[a_{i,j}] = A \in \mathbb{R}^{n \times m}$ be a real matrix that represents feature map region which is being passed to a pooling function. Most commonly used pooling methods include:

- **Max pooling**, which replaces the input region with its maximum value:

$$p(A) = max(A) = max(\{a_{i,j} | i \in 1, ..., n, j \in 1, ..., m\}). \tag{4.5}$$

- **Average pooling** and **weighted average pooling** aggregate the input region by taking its average or a weighed sum, which can be based on the distance from the region's center:

$$p(A) = \frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} a_{i,j}. \tag{4.6}$$

- **L_2** pooling computes the L_2 norm of the vector constructed by "unfolding" the input region.

$$p(A) = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m} a_{i,j}^2}. \tag{4.7}$$

(a) : Original image. **(b) :** Pooled image.

Figure 4.2: Illustration of the 2x2 max pooling function. Each region of the original image of size 2x2 was replaced by its maximum value.

Pooling is sometimes considered to be a separate layer, but in most popular architectures it always follows convolution and can be considered as its part. However, it can be situational and there are successful architectures which use pooling following several stacked convolution operations or which do not use pooling at all [45]. It can be adjusted with the padding and stride hyperparameters, analogous to convolution.

Pooling is used to reduce the amount of variables in the model, thus preventing overfitting and improving its computational efficiency. Although heavy use of pooling can lead to agressive information loss and underfitting, so it should be treated lightly. Another purpose of using a pooling function is introducing *translational invariance* into the model [15]. By considering whole regions instead of separate input values, pooling helps to put a bigger emphasis on the feature's value regardless of its location. As a result, the network becomes resistant to small perturbations of the input.

4.2.3 Batch Normalization

Batch normalization layer [24] normalizes the output of the layer before it. Let $\mathbf{X} \subseteq \mathbb{D}$ be a batch of inputs, then:

Algorithm 4.1 Compute the output $\mathbf{y}$ of the batch normalization layer.

$\mu \leftarrow \frac{1}{|\mathbf{X}|}\sum_{\mathbf{x}\in\mathbf{X}} \mathbf{x}$
$\sigma^2 \leftarrow \frac{1}{|\mathbf{X}|}\sum_{\mathbf{x}\in\mathbf{X}} (\mathbf{x}-\mu)^2$
$\hat{\mathbf{x}} \leftarrow \frac{\mathbf{x}_i-\mu}{\sqrt{\sigma^2+\epsilon}}$
$\mathbf{y} = \gamma\hat{\mathbf{x}} + \beta$ ▷ Scale and bias

By centering and scaling the feature maps batch normalization makes the gradient computation more robust because. The main goal of the layer is

to discard the change in the distribution of hidden layer outputs, which can happen in the process of training. This simplifies learning and provides faster convergence toward the minumum. The values of $\gamma \in \mathbb{R}$ and $\beta \in \mathbb{R}$ are determined in the process of learning.

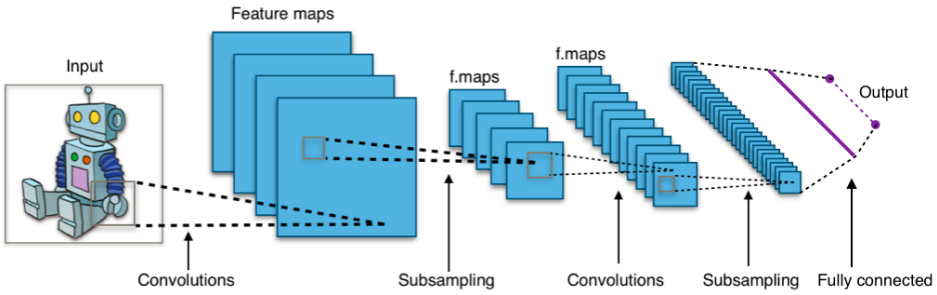


Figure 4.3: CNN illustration [2]. It follows the typical CNN architecture pattern: convolutional layers are followed by subsampling, e.g. max pooling, and the width of the model increases as the depth grows.

4.3 Main Concepts Behind CNNs

This section describes most important ideas behind CNNs and tries to explain why they improve a normal ANN system.

4.3.1 Local Receptive fields

In traditional fully-connected layers every output is connected with every input. In CNN, neuron connections are restricted only to the subset of adjacent inputs through kernel. This subset of neurons is sometimes referred to as neuron's **local receptive field**.

This approach is aimed at the detection of local patterns and small details in the input.

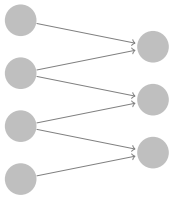


Figure 4.4: Local receptive fields. Sparsely connected neurons receive information only from a subset of predecessor units.

Despite the typically small size of the receptive field, neurons in located in the deeper layers indirectly interact with the larger area of the input. By the nature of the convolution, if layer's output is defined to be smaller than the input, its feature map will consist of aggregated input regions. Following

this principle, feature maps in the deepest layers of the network will assemble "dense" information describing the interactions between different regions if the input.

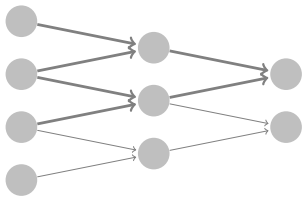


Figure 4.5: Deep indirect interactions. The first unit of the right-most layer is directly connected only to two units of the previous layer, but it also receives the data from the first layer through them.

4.3.2 Parameter Sharing

One of the most substantial downsides of the multilayer perceptron is large increase in the number of trainable parameters with each new layer added to the model, which makes models impractical to train and restricts the width and the depth of the network. In the meantime, convolutional layers have far less parameters that are shared across the whole layer.

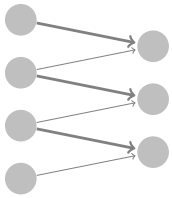


Figure 4.6: Parameter sharing. All neurons in the layer have the same number of parameters which are shared between them. There are only two parameters in the network of the figure, they are distinguished with the connection thickness.

Let m, n be the number of neurons in two consecutive layers, resp. Then traditional fully-connected layer would have $n(m+1)$ trainable parameters (with the addition of biases). But convolutional layer with the kernel size of $w \times h$ ends up with the total of $wh + n$ weights and biases. This number can be sufficiently less than that of the fully-connected layer.

Thus, parameter sharing results in a smaller number of model parameters and positively influences computational and storage costs during the training and prevents overfitting. However, it is worth noting that in some problems it is reasonable to have unshared parameters. For instance, in the problem of the face recognition where the dataset contains centered and normalized

images different filters would capture different information, although at a greater computational cost.

4.4 Popular CNN Architectures

- LeNet [31]. The first successful applications of CNNs. LeNet architecture was used to read zip codes, digits, etc.
- AlexNet [30]. The first work that popularized CNNs for Computer Vision was the AlexNet. The AlexNet was submitted to the ImageNet ILSVRC [1] challenge in 2012 and took the first place with the top 5 error of 16% compared to the second place with 26% error. The network featured multiple convolutional layers stacked on top of each other without the intermission of the pooling layers.
- GoogLeNet [49]. The ILSVRC 2014 winner. Its main contribution was the development of an Inception Module. Inception module uses convolutional filter of different sizes and concatenates their outputs, letting the model decide which filter size does the best job at capturing features. Additionally, it uses Average Pooling instead of Fully Connected layers at the final network layers, eliminating a large amount of parameters which arises in fully-connected layers.
- VGGNet [43]. The runner-up in ILSVRC 2014 was the network that became known as the VGGNet. It showed the importance of the network depth. The network contains 16 layers (without pooling) and only performs 3x3 convolutions and 2x2 pooling. However, VGGNet is quite expensive to evaluate and uses a lot of memory for its 140 millions of parameters.
- ResNet [20]. Residual Network was the winner of ILSVRC 2015. It features special residual blocks that improve the optimization of the cost function and a heavy use of batch normalization. The architecture also rejected the pattern of using fully connected layers at the end of the network.
- Squeeze-and-Excitation Network [22] was the ImageNet 2017 winner. It introduced the mechanism of adaptive weighting of intermediate feature map channels in the hidden layers of the network, thus increasing the ability of the network to learn best representations of the input.

As it can be observed from this list, the field of CNNs is rapidly changing. The best improvements of CNN architecture are often based on simple ideas, but are very effective.

[1] http://www.image-net.org/challenges/LSVRC/

Chapter 5

Experimental Results

5.1 MNIST

5.1.1 Data Description

The MNIST database [54] (Modified National Institute of Standards and Technology database) contains centered and size-normalized hand-written digits. It consists of 28x28 one-channel gray-scale images. The training set has 60000 images and the test set has 1000 images. It is a popular benchmark dataset for those who would like to test a computer vision algorithm without preprocessing the data or a good starting point for the beginners in the area of CNN.

The task is to classify an input image to one of the 10 classes representing numerical digits.

Figure 5.1: MNIST dataset digits.

5.1.2 Experimental setup

No data preprocessing was done, and the following architecture was used:

Layer type	Layer parameters
Input	shape: 28x28x1
Convolution	kernel shape: 5x5x1x32, padding: 1, strides: 1, ReLU activation
Max pooling	window shape: 2x2, padding: 1, strides: 1
Convolution	kernel shape: 5x5x32x64, padding: 1, strides: 1, ReLU activation
Max pooling	window shape: 2x2, padding: 1, strides: 1
Fully-connected	1024 units, ReLu activation
Dropout	keep probability: 0.65
Output	10 units, softmax transformation

Table 5.1: CNN architecture used for MNSIT digit classification

Learning was conducted in the following setting:

Training parameter	Value
Number of epochs	25
Batch size	128
Optimizer	Adam: $\beta_1 = \beta_2 = 0.999$, $\epsilon = 10^{-8}$
Initial learning rate	10^{-3}
Weight initialization method	Truncated normal distribution

Table 5.2: Learning parameters used for MNSIT digit classification

It is a pretty standard architecture for this problem. Depth of the model is low, but is sufficient to fit the data. Adam optimizer effectively minimizes cost function without any parameter tuning. Dropout is applied to improve generalization and performance on the test set. Convolutional layers are always followed by the pooling layers, this limits the capabilities of this network due to the aggressive information loss in pooling.

5.1.3 Learning Results

After each epoch the model was evaluated on the validation dataset, which contains 10000 images, as in the test one. Results are presented below. The model achieved **99.1%** accuracy on the test set.

Figures (5.2) and (5.3) illustrate progression of training and validation loss, respectively. Fluctuations in the training loss function illustrate the typical behavior of stochastic gradient algorithm.

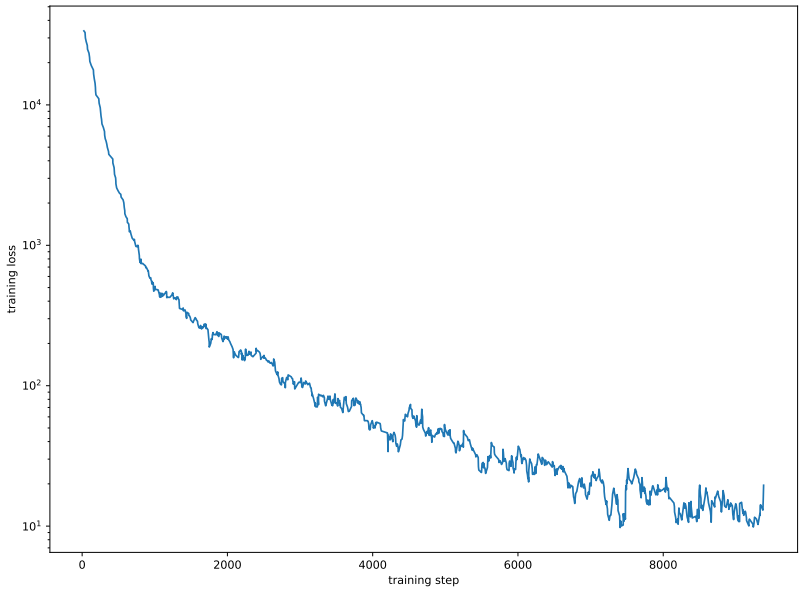


Figure 5.2: Training loss progression for MNIST dataset experiment.

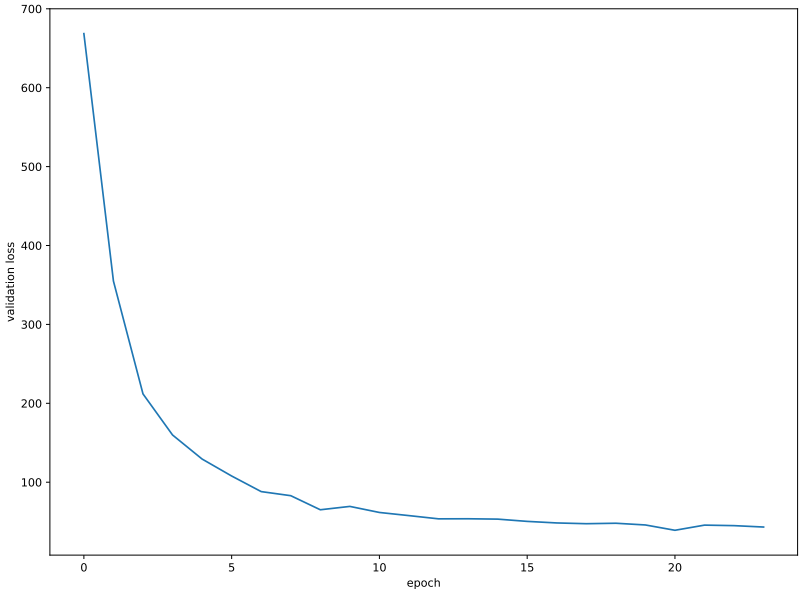


Figure 5.3: Validation loss progression for MNIST dataset experiment.

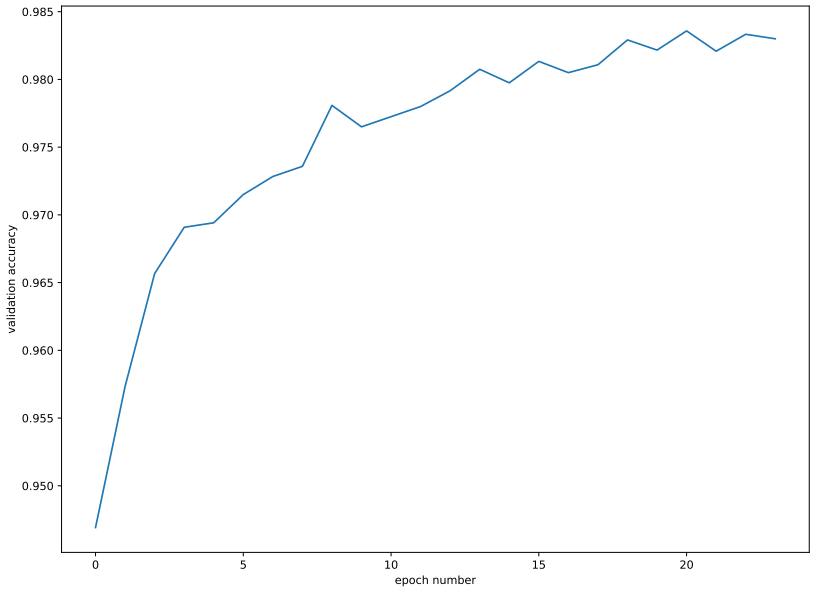


Figure 5.4: Validation accuracy progression for MNIST dataset experiment.

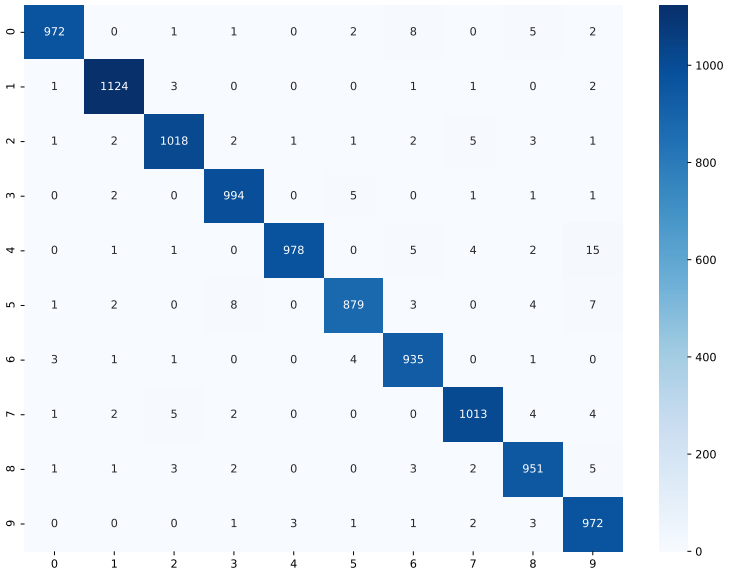


Figure 5.5: MNIST test set confusion matrix. The number of confused fours and nines can be explained by the visual similarity of these digits and different writing styles of the experiment participants.

5.1.4 Discussion

Model	result (accuracy, %)
CNN with DropConnect [51]	99.79
CNN LeNet-5, data augmentation [31]	99.2
Experiment CNN	**99.1**
CNN LeNet-1 [31]	98.3
3-layer NN, 500+150 HU, data augmentation [31]	97.55

Table 5.3: A comparison of selected MNIST architectures. The model on the first line is currently state-of-the art for this dataset.

As it can be observed, the experimental model performance stands in the middle between deep CNNs and smaller ones along with plain MLPs. It is an expected result from such small and shallow network, as achieving better accuracy requires using more sophisticated methods.

5.2 CIFAR-10

5.2.1 Data Description

The **CIFAR-10** dataset [28] consists of 60000 32x32 color images from 10 classes, with 6000 images per class. It is divided into 50000 training images and 10000 test images. Each image consists of 3 channels according to the standard RGB color scheme. One of the challenged posed by this dataset is a relatively small size of images considering the complexity of objects they are depicting.

The task is to classify images between 10 classes:

1. Airplane
2. Automobile
3. Bird
4. Cat
5. Deer
6. Dog
7. Frog
8. Horse
9. Ship
10. Truck

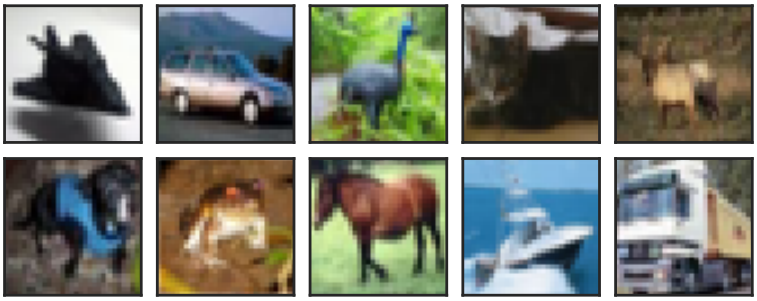


Figure 5.6: CIFAR-10 dataset image examples.

5.2.2 Experimental setup

Common data augmentation techniques were used to increase the size of the training dataset. Random flipping, whitening and blurring were applied

independently with 0.5 chance of each transformation. Resulting training set contains 100000 randomly mixed up original and augmented images.

The task was approached with the following CNN architecture:

Layer type	Layer parameters
Input	shape: 32x32x3
Convolution	kernel shape: 5x5x3x64, padding: 1, strides: 1, ReLU activation
Batch normalization	–
Max pooling	window shape: 3x3, padding: 2, strides: 2
Convolution	kernel shape: 5x5x64x64, padding: 1, strides: 1, ReLU activation
Batch normalization	–
Max pooling	window shape: 3x3, padding: 2, strides: 2
Fully-connected	256 units, ReLu activation
Output	10 units, softmax transformation

Table 5.4: CNN architecture used for CIFAR-10 image classification.

Learning was conducted in the following setting:

Learning parameter	Value
Number of epochs	100
Batch size	128
Optimizer	Adagrad
Initial learning rate	$\alpha = 10^{-2}$
Loss regularization method	L_2, $\lambda = 10^{-2}$
Weight initialization method	Truncated normal distribution, $\sigma^2 = 0.05$

Table 5.5: Learning parameters used for CIFAR-10 digit classification.

This architecture relies on batch normalization and L_2 loss to achieve generalization. Adagrad and batch normalization accelerate convergence to the minimum, while methods with momentum tend to oscillate too much.

5.2.3 Learning Results

After each epoch the model was evaluated on the validation dataset, which contains 10000 images. Results are presented below. The model achieved **76.65%** accuracy on the test set.

Figure (5.7) shows the change in the training dataset loss. Notice a few sudden peaks, which are also registered on the (5.8) and (5.9).

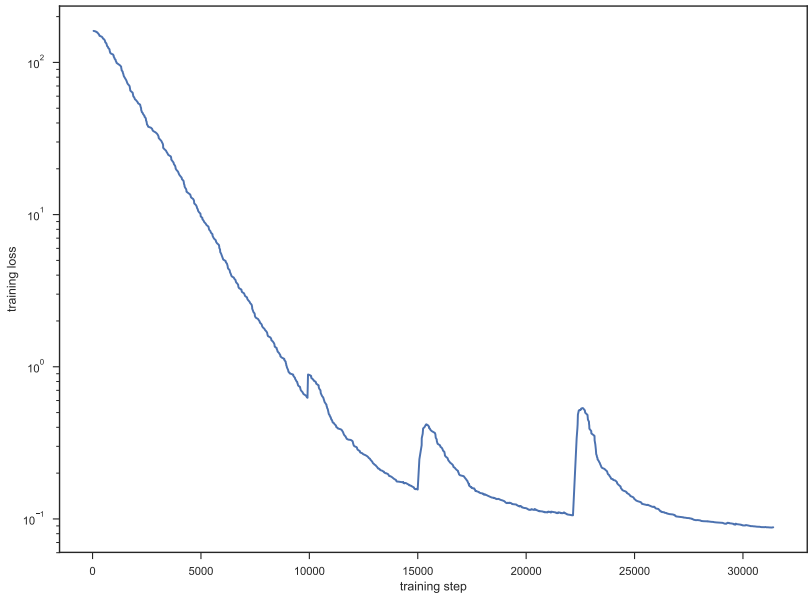


Figure 5.7: Training loss progression for CIFAR-10 dataset experiment.

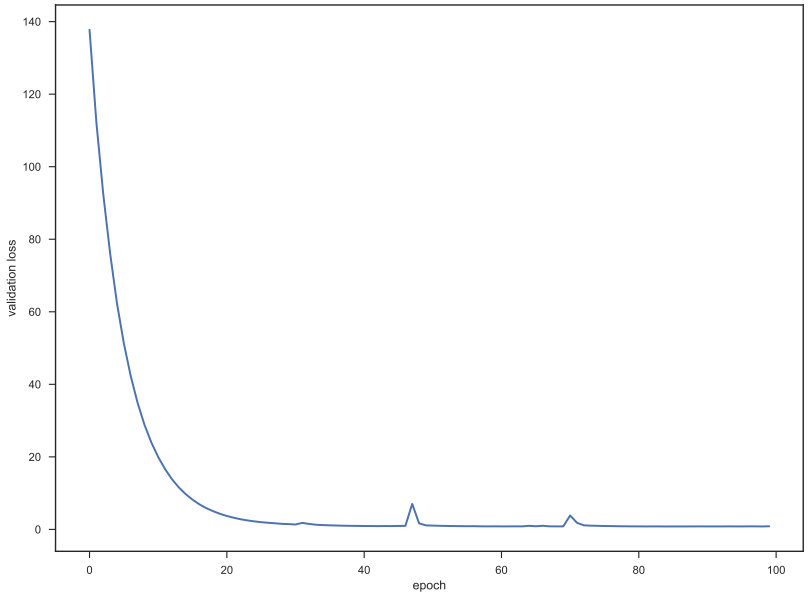


Figure 5.8: Validation loss progression for CIFAR-10 dataset experiment.

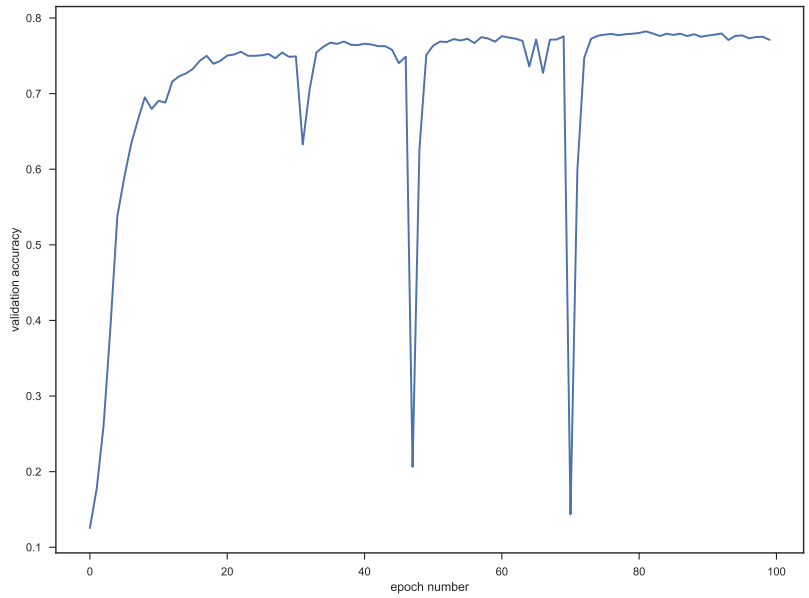


Figure 5.9: Validation accuracy progression for CIFAR-10 dataset experiment.

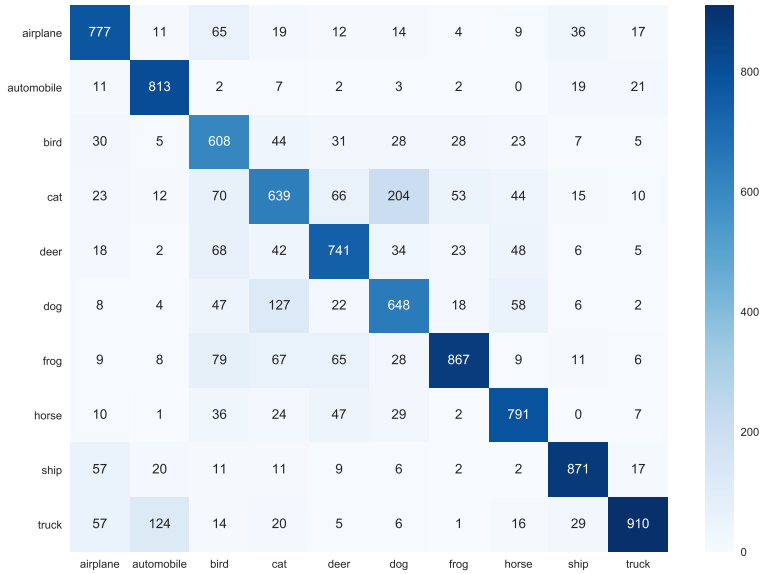


Figure 5.10: CIFAR-10 test set confusion matrix. Most common mistakes are confusing dogs with cats and birds with planes (and vice versa), which naturally corresponds to their real visual appearance. This could be a hint that there is no overfitting and model generalizes well, albeit underperforming.

5.2.4 Discussion

This architecture is similar to the architecture chosen for the MNIST dataset, yet it is able to perform adequately on the harder problem due to the batch normalization.

Model	result (accuracy, %)
Deep conv net, fractional max-pooling [16]	96.53
Deep conv net, leaky ReLU [53]	88.8
Experimental CNN	**76.65**
Fast-Learning Shallow Convolutional Neural Network [34]	75.86

Table 5.6: A comparison of selected CIFAR-10 architectures. The model on the first line is currently state-of-the art for this dataset.

5.3 Recurrence Plots of Electroencephalogram Signals

EEG is an electrophysiological method of recoding brain activity. Is is based on measuring temporal changes in electromagnetic potential within different brain areas [36].

The original EEG dataset was recieved from National Institute of Mental Health in Czech Republic [1], and it featured 19-channel EEG measurements recorded from 7 persons performing suffering from some kind of mental illness or not. The dataset was then transformed in order to increase its dimensionality. Individual EEG channels were sliced into parts of the length equal to 256, and these subseries are transformed into two-dimensional one-channel images using recurrence plot technique. Let $\mathbf{s} : \mathbb{Z} \to \mathbb{R}$ be the discretized signal in a form of a time series. Let the **Recurrence Plot** of this signal be a matrix of distances between individual signal's values:

$$I(i, j) = |\mathbf{s}(i) - \mathbf{s}(j)|. \tag{5.1}$$

Recurrence plots are successfully applied to the analysis of dynamical systems [10]. They are able to highlight recurring patterns of the signal and represent them in a two-dimensional form which is suitable for analysis by a CNN.

Resulting dataset contains 9291 images of shape 256x256x1 labeled with positive and negative class telling whether the person is performing any activity or not. The task is binary classification of the images.

[1] http://www.nudz.cz/en/

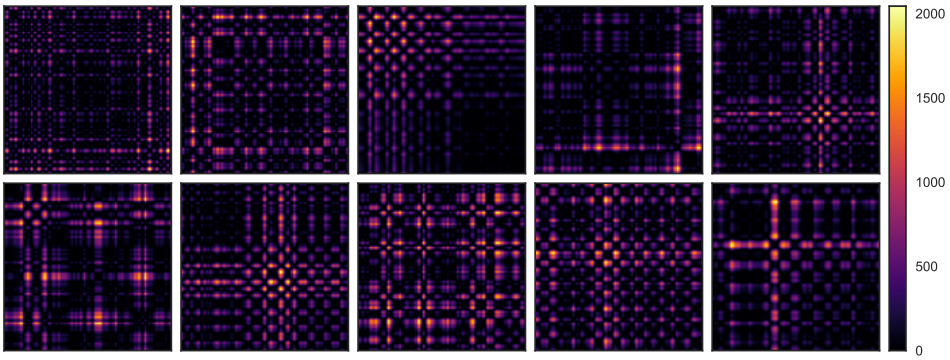


Figure 5.11: Recurrence plots of EEG examples.

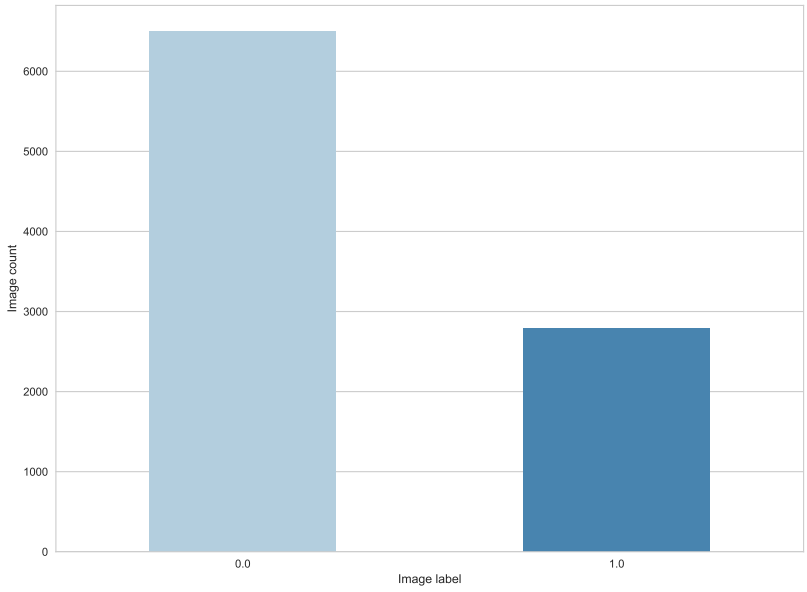


Figure 5.12: Class imbalance in the EEG dataset.

The dataset is slightly imbalanced, since the ratio between positive and negative samples is approximately 0.4. This can cause the classifier to stop in a local minimum, always predicting the inputs as negative. In the case of SGD and its modifications it is important to keep the data shuffled, so that no batch or consequential batches contain samples of only one class instance.

5.3.1 Experimental setup

Before the learning phase, images were normalized by mean-centering following with standard deviation scaling:

$$I(i,j) = \frac{I(i,j) - \mu_{i,j}}{\sigma_{i,j}}, \tag{5.2}$$

where $\mu_{i,j}$ and $\sigma_{i,j}$ is the mean and standard deviation values of the pixel across all images, respectively.

The task was approached with the following model:

Layer type	Layer parameters
Input	shape: 256x256x1
Convolution	kernel shape: 3x3x1x16, padding: 1, strides: 1, ReLU activation
Convolution	kernel shape: 3x3x16x16, padding: 1, strides: 1, ReLU activation
Max pooling	window shape: 2x2, padding: 1, strides: 1
Dropout	keep probability: 0.75
Convolution	kernel shape: 3x3x16x32, padding: 1, strides: 1, ReLU activation
Convolution	kernel shape: 3x3x32x32, padding: 1, strides: 1, ReLU activation
Max pooling	window shape: 2x2, padding: 1, strides: 1
Dropout	keep probability: 0.75
Fully-connected	128 units, ReLu activation
Dropout	keep probability: 0.5
Output	1 unit, sigmoid transformation

Table 5.7: CNN architecture used for recurrence plots classification.

Learning was done with in the following setting:

Learning parameter	Value
Number of epochs	35
Batch size	64
Optimizer	Adam: $\beta_1 = \beta_2 = 0.999$, $\epsilon = 10^{-8}$
Initial learning rate	10^{-4}
Weight initialization method	Truncated normal distribution, $\sigma^2 = 0.1$
Random seed	23

Table 5.8: Learning parameters used for EEG data classification.

The amount of available training data is much smaller than that in benchmark problems, so the number of filters in each layer is also chosen to be smaller in an attempt of building the network as deep and as simple as possible. The increased usage of dropout contributes to the model simplicity as well. This model makes use of two stacked convolutional layers to achieve better deep interactions.

5.3.2 Learning Results

This architecture achieved 81.7% accuracy, 80.6% of precision and 52.62% of recall on the test set for the binary classification threshold equal to 0.5.

Figures (5.13) to (5.17) show the change in the different metrics as the optimization algorithm progresses. All of those metrics are rather noisy, but the trend is always positive. In the case of precision (5.16) it might look like it does not getting much better, but its variance is decreased with each iteration and it converges to the value of 0.8.

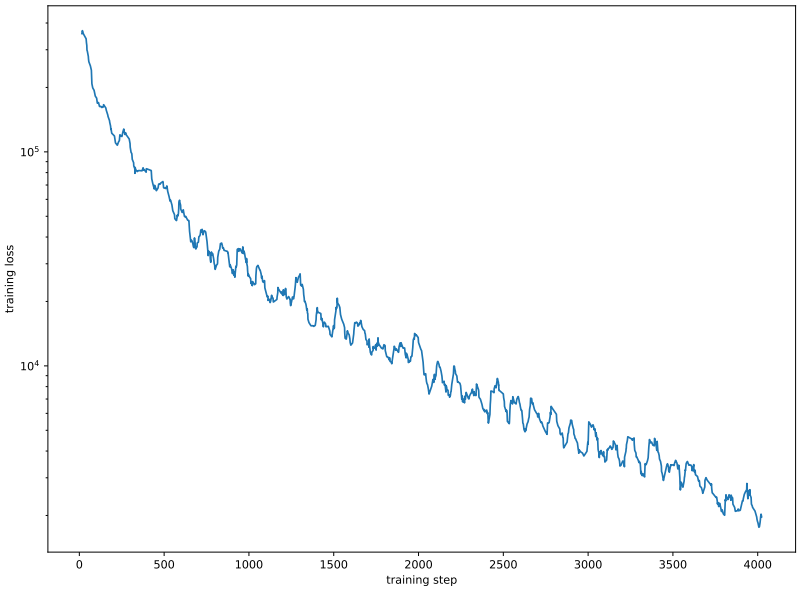


Figure 5.13: Training loss progression for EEG dataset experiment.

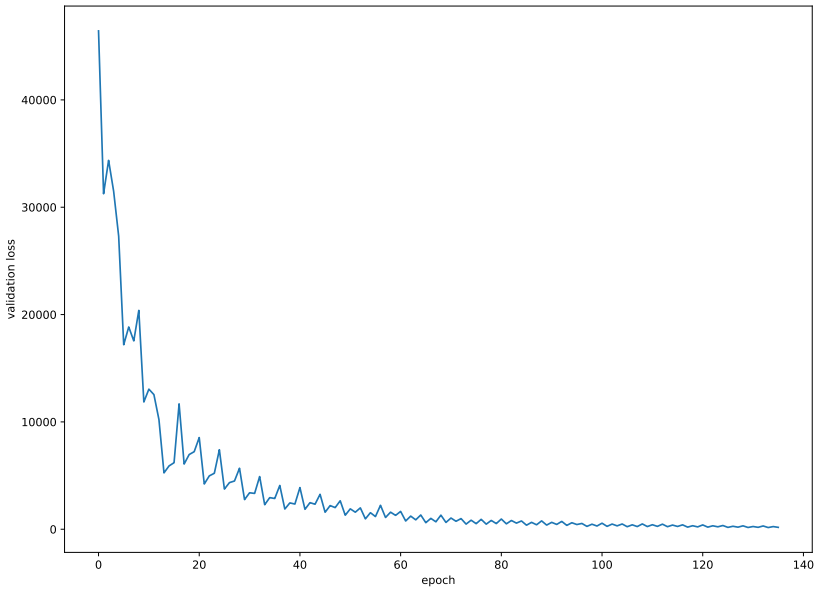


Figure 5.14: Validation loss progression for EEG dataset experiment.

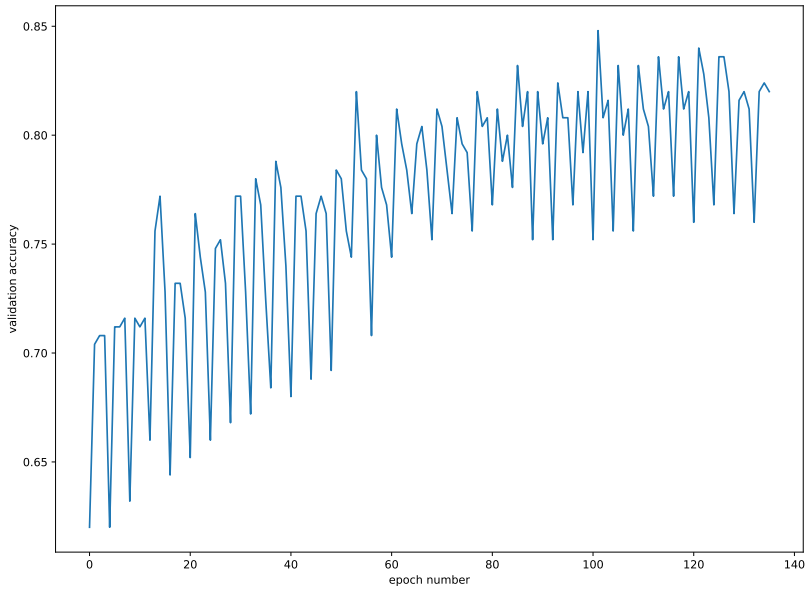


Figure 5.15: Validation accuracy progression for EEG dataset experiment.

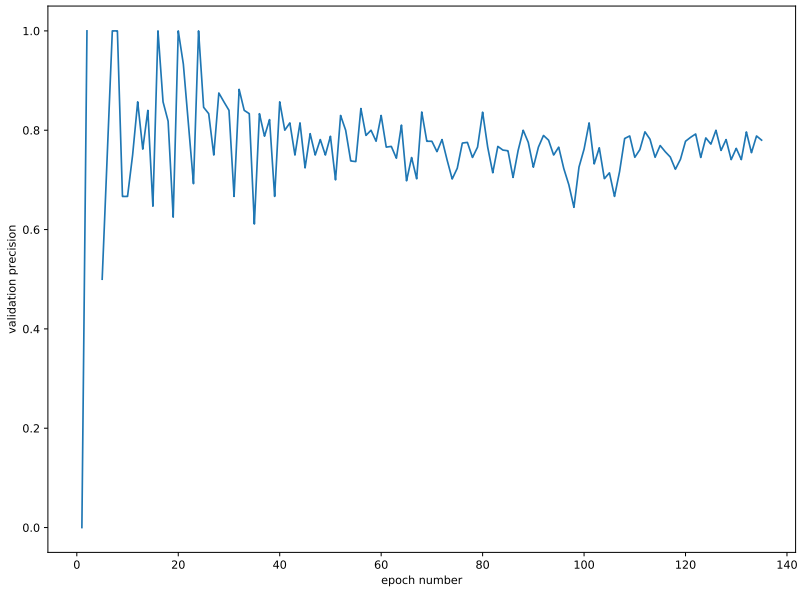


Figure 5.16: Validation precision progression for EEG dataset experiment.

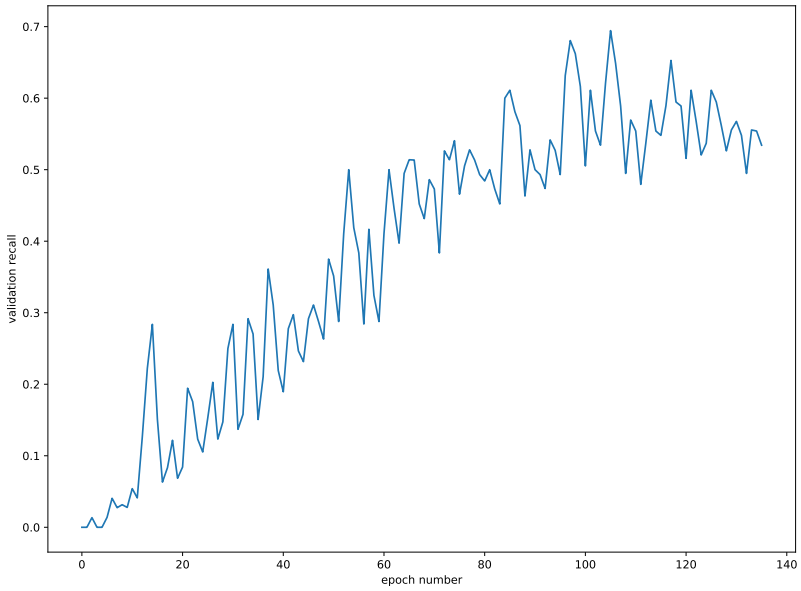


Figure 5.17: Validation recall progression for EEG dataset experiment.

5.3.3 Discussion

This architecture achieved reasonable performance, although the training process is a bit unstable, possibly because of the class imbalance which may lead to unevenly represented classes in different batches.

Chapter 6

Conclusions

This thesis contains a theoretical overview of CNNs, as well as an experimental analysis of their performance on popular benchmark problems and real world data.

In this text we showed how CNNs are an improvement on the general idea of artificial neural networks. Introduction of *parameter sharing* and *local connectivity* made very deep CNNs not only feasible, but very efficient. Their ability to automatically learn appropriate representations of the data makes problems involving large amounts of data that would otherwise require a lot of preprocessing easier to solve. As the retrospective analysis of state-of-the-art models shows, the field of CNNs is rapidly changing. Most innovative CNN architectures are based on simple yet powerful ideas, and their elegant mathematical implementation makes the architecture successful.

The performance of CNN was evaluated in two kind of problems: manually-created benchmark datasets and real-world datasets. Two benchmark dataset were chosen for experiments. First one is the MNIST database of hand-written digits, providing a problem of classifying input into one of the classes corresponding to a digit. We achieved a reasonable performance with a shallow network, due to the power of the Adam optimization algorithm. Besides, a CNN similar to the first one was applied on a much harder problem – CIFAR-10 dataset of small images. It has achieved a basic performance in a relatively small number of steps, making use of batch normalization technique. Finally, the real-world dataset is a collection of EEG signals transformed into two-dimensional recurrence plots. It poses a problem of classification recurrence plots into two classes: as produced by mentally healthy person or by a person having some kind of mental disease. The proposed architecture reached an accuracy of around 80% and maintained a precision on the same level. We can conclude that the CNN is powerful ML tool for classifying EEG signals and images. Although, more could be done to improve its recall measure, which stayed at the level of 50%. E.g. eliminating class imbalance with oversampling technique.

In future works we would like to evaluate a CNN over functional Magnetic Resonance Imaging (fMRI) data. Also we would like to conduct more experiments concerning CNN architecture tuning. Finally, it may be concluded that this thesis was an excellent opportunity to improve our knowledge in

supervised learning, classification problems and in the popular CNN model.

Acronyms

Adagrad Adaptive gradient. 17

Adam Adaptive moment estimation. 17

ANN Artificial Neural Network. 1, 2, 5, 11, 13, 15, 16, 20, 21, 23, 26

BP Back-propagation. 18

CNN Convolutional Neural Network. vi, viii, ix, 1, 2, 21–23, 26, 28, 29, 33, 35, 38, 40, 45

DL Deep Learning. 1, 2, 6, 11, 14, 21, 23

EEG Electroencephalography. vi, viii, ix, 1, 2, 38–43

fMRI functional Magnetic Resonance Imaging. 1, 45

GD Gradient Descent. 16

GPU Graphics Processing Unit. 1

ML Machine Learning. 1–5, 15, 21, 45

MLP Multilayer perceptron. 14, 18, 19, 33

PR curve Precision-Recall curve. viii, 8, 9

ROC curve Receiver Operating Characteristic curve. viii, 8

SGD Stochastic Gradient Descent. 16, 17, 39

Bibliography

[1] David M Allen. Mean square error of prediction as a criterion for selecting variables. *Technometrics*, 13(3):469–475, 1971.

[2] Aphex34. Typical cnn illustration. `https://commons.wikimedia.org/wiki/File:Typical_cnn.png`.

[3] Sylvain Arlot, Alain Celisse, et al. A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79, 2010.

[4] C.M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 2006.

[5] C.M. Bishop. *Pattern Recognition and Machine Learning*, chapter Linear Models for Regression. Information Science and Statistics. Springer, 2006.

[6] Dan Ciresan, Alessandro Giusti, Luca M. Gambardella, and Jürgen Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2843–2851. Curran Associates, Inc., 2012.

[7] Dan C Cireşan, Alessandro Giusti, Luca M Gambardella, and Jürgen Schmidhuber. Mitosis detection in breast cancer histology images with deep neural networks. In *International Conference on Medical Image Computing and Computer-assisted Intervention*, pages 411–418. Springer, 2013.

[8] Bin Dai, Shilin Ding, Grace Wahba, et al. Multivariate bernoulli distribution. *Bernoulli*, 19(4):1465–1483, 2013.

[9] Tim Dettmers. Deep learning in a nutshell: Core concepts. `https://devblogs.nvidia.com/deep-learning-nutshell-core-concepts/`.

[10] J-P Eckmann, S Oliffson Kamphorst, and David Ruelle. Recurrence plots of dynamical systems. *EPL (Europhysics Letters)*, 4(9):973, 1987.

[11] Michael Egmont-Petersen, Dick Ridder, and Heinz Handels. Image processing with neural networks - a review. pattern recogn 35:2279c2301. 35:2279–2301, 10 2002.

[12] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115, 2017.

[13] David A Fournier, Hans J Skaug, Johnoel Ancheta, James Ianelli, Arni Magnusson, Mark N Maunder, Anders Nielsen, and John Sibert. Ad model builder: using automatic differentiation for statistical inference of highly parameterized complex nonlinear models. *Optimization Methods and Software*, 27(2):233–249, 2012.

[14] Yoav Goldberg. A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57:345–420, 2016.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[16] Benjamin Graham. Fractional max-pooling. *arXiv preprint arXiv:1412.6071*, 2014.

[17] A. Graves, A. r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, May 2013.

[18] K. Gurney. *An Introduction to Neural Networks*. UCL Press, London, 1997.

[19] S. Haykin and S.S. Haykin. *Neural Networks and Learning Machines*. Number v. 10 in Neural networks and learning machines. Prentice Hall, 2009.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[21] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251 – 257, 1991.

[22] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 2017.

[23] David H Hubel and Torsten N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574–591, 1959.

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

[25] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[26] Yoram Singer John Duchi, Elad Hazan. *Journal of Machine Learning Research*, pages 2121–2159, Jul 2011.

[27] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[28] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[31] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[32] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–444, 2015.

[33] Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.

[34] Mark D McDonnell and Tony Vladusich. Enhanced image classification with a fast-learning shallow convolutional neural network. In *Neural Networks (IJCNN), 2015 International Joint Conference on*, pages 1–7. IEEE, 2015.

[35] Marvin L. Minsky and Seymour A. Papert. *Perceptrons: Expanded Edition*. MIT Press, Cambridge, MA, USA, 1988.

[36] Ernst Niedermeyer and FH Lopes da Silva. *Electroencephalography: basic principles, clinical applications, and related fields*. Lippincott Williams & Wilkins, 2005.

[37] Kyoung-Su Oh and Keechul Jung. Gpu implementation of neural networks. *Pattern Recognition*, 37(6):1311 – 1314, 2004.

[38] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145 – 151, 1999.

[39] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain [j]. 65:386 – 408, 12 1958.

[40] S.J. Russell, S.J. Russell, P. Norvig, and E. Davis. *Artificial Intelligence: A Modern Approach*. Prentice Hall series in artificial intelligence. Prentice Hall, 2010.

[41] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 44(1.2):206–226, Jan 2000.

[42] John Shore and Rodney Johnson. Properties of cross-entropy minimization. *IEEE Transactions on Information Theory*, 27(4):472–482, 1981.

[43] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[44] Jan A Snyman. Practical mathematical optimization. 2005.

[45] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[46] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[47] Daniel Strigl, Klaus Kofler, and Stefan Podlipnig. Performance and scalability of gpu-based convolutional neural networks. In *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, pages 317–324. IEEE, 2010.

[48] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[49] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, et al. Going deeper with convolutions.

[50] Mitchell T.M. *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill, 1997.

[51] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1058–1066, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[52] Bernard Widrow and Marcian E. Hoff. Adaptive switching circuits. In *1960 IRE WESCON Convention Record, Part 4*, pages 96–104, New York, 1960. IRE.

[53] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[54] Christopher J.C. Burges Yann LeCun, Corinna Cortes. the mnist database of handwritten digits. `http://yann.lecun.com/exdb/mnist/`.

[55] Masood Zamani and Stefan C Kremer. Neural networks in bioinformatics. In *Handbook on Neural Information Processing*, pages 505–525. Springer, 2013.