

Czech Technical University in Prague  
Faculty of Electrical Engineering  
Department of Computer Science



Master`s Thesis

NEURAL NETWORKS USING FOR HANDWRITTEN NUMBERS RECOGNITION

Dina Latypova

Supervisor: **Ing. Karel Frajták, Ph.D.**

Study Program: Open Informatics  
Field of Study: Software Engineering

May 22, 2020



## I. Personal and study details

Student's name: **Latypova Dina** Personal ID number: **492136**  
Faculty / Institute: **Faculty of Electrical Engineering**  
Department / Institute: **Department of Computer Science**  
Study program: **Open Informatics**  
Specialisation: **Software Engineering**

## II. Master's thesis details

Master's thesis title in English:

**Neural networks using for handwriting numbers recognition**

Master's thesis title in Czech:

**Využití neuronové sítě pro rozpoznávání ručně psaných číslic**

Guidelines:

Build a neural network that will recognize images of handwritten numbers. The neural network consists of the Kohonen neural network, which defines the main clusters for the entire database, and the Hopfield neural network, which identifies the closest image for a particular class. This neural network presents 2 methods for identifying the most "close" image. Images are compared as a percentage of black and white pixel matches.

Bibliography / sources:

[1] X. Zhang, X. Zhou, M. Lin, J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," Proceedings IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 6848–6856. ; [2] D.N. Tumakov, D.M. Khairullina, A.A. Valeeva, "Recovery of parameters of a homogeneous elastic layer using neural networks," Journal of Fundamental and Applied Sciences, vol. 9, 2017, pp. 1202–1220.

Name and workplace of master's thesis supervisor:


**Ing. Karel Frajták, Ph.D., Software Testing Intelligent Lab, FEE**

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

Date of master's thesis assignment: **12.03.2020** Deadline for master's thesis submission: **22.05.2020**

Assignment valid until: **19.02.2022**

  
Ing. Karel Frajták, Ph.D.  
Supervisor's signature

  
Head of department's signature

  
prof. Mgr. Petr Páta, Ph.D.  
Dean's signature

## III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the master'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 sincere gratitude to my supervisor Ing. Dmitrii Tumakov (KFU) for guidance and immense help on my thesis. He was always ready to offer the right and appropriate solutions whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed this paper to be my own work, but steered me in the right direction whenever they thought I needed it.

Also I want to appreciate Turilova Ekaterina, Enikeev Arslan, Miroslav Bureš and all Czech team for the opportunity to participate in the Double Degree program (CVUT, KFU). This chance opened up new perspectives for me and my future investigations in this area.

Finally, I want to especially thank my family, friends and colleagues who prop up my morale throughout my thesis work. I really appreciate them providing me with unfailing support and continuous encouragement during my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them



# Declaration

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publications used.

I have no objection to usage of this work in compliance with the act §60 Zákon č. 121/2000Sb. (copyright law), and with the rights connected with the copyright act including the changes in the act.

Prague, May, 2020

---





# Abstract

LATYPOVA, Dina: Neural networks using for handwriting numbers recognition. [Master's Thesis] - Czech Technical University in Prague. Faculty of Electrical Engineering, Department of Computer Science. Supervisor: Ing. Karel Frajták, Ph.D.

In the presented work, a Hopfield neural network was constructed for recognizing handwritten digit patterns contained in the MNIST database. Ten Hopfield neural networks were built for each digit separately. The centers of clusters that were built using the Kohonen neural network were taken as objects for "memorization". Two methods were proposed, which are a supported step in a Hopfield neural network; an analysis of these methods was carried out. Also, an error was calculated for each method, the pros and cons of their use were identified.

Clustering of handwritten digits from the training sample of the MNIST database is conducted. Clustering is performed using a Kohonen neural network. The optimal number of clusters (not exceeding 50) for each digit is selected. As a metric for Kohonen network, the Euclidean norm is used. The network is trained by a serial algorithm on the CPU and by a parallel algorithm on the GPU using CUDA technology. The graphs of the time spent on training the neural network for each digit are given. A comparison of the time spent for serial and parallel training is presented. It is found that the average value of accelerating the training of a neural network using CUDA technology is almost 17-fold. The digits from the test sample of the MNIST database are used to evaluate the accuracy of building the cluster. It is found that the percentage of vectors from the test sample in the correct cluster for each digit is more than 90%. The F-measure for each digit is calculated. The best values of the F-measure are obtained for 0 and 1 (F-measure is 0.974), whereas the worst values are obtained for the digit 9 (F-measure is 0.903).

The introduction briefly describes the content of the work, what research is currently available, and the relevance of this work. This is followed by a statement of the problem, as well as what technologies were used to write this work. The first chapter describes the theoretical aspects, as well as describes how to solve each stage of this work. The second chapter contains a program description of the work and the results obtained. In the second chapter, we talk about parallelizing the learning algorithm of the Kohonen neural network. In the third chapter, the software is tested.

The results are the recognition response of each neural network - the image is the most similar to the image submitted for input, also, the total percentage of recognition for each neural network.

**Keywords:** Hopfield neural network, Kohonen neural network, handwritten digits, MNIST database, Parallelization, CUDA .



# Contents

Introduction	1
Thesis content	2
Chapter 1	3
Neural networks and MNIST database.	3
1.1 Artificial Neuron and Artificial Neural Networks	3
1.2 Classification and Training of Neural Networks	6
1.3 The Kohonen Neural Network. SOM.	7
1.4 Clustering Methods	9
1.5 Automatic Clustering Methods	9
1.6 Recurrent Neural Networks	10
1.7 Overview of Neural Networks for Pattern Recognition	13
1.8 MNIST database	15
1.9 Using CUDA Technology for Neural Network	16
Chapter 2	18
Description of the software implementation and results	18
2.1 Task Statement and Program Structure	18
2.2 Determining the Optimal Numbers of Clusters	27
2.3 Data Clustering and Cluster Analysis	31
2.4 Pattern Recognition by a Hopfield Neural Network	41
2.5 Parallel Implementation of Kohonen Neural Network training	47
Chapter 3	54
Testing	54
3.1 Testing on the Fashion MNIST dataset	54
Conclusion	60
Bibliography	61

# List of figures

Fig. 1.1: Neuron .....	4
Fig. 1.2: Rosenblatt's Perceptron.....	4
Fig. 1.3: Artificial neuron.....	4
Fig. 1.4: Architecture of the Kohonen neural network .....	7
Fig. 1.5: The feedback neuron.....	11
Fig. 1.6: Architecture of Hopfield neural network .....	11
Fig. 1.7: Architecture of a standard recurrent neural network .....	14
Fig. 1.8: Architecture of a LSTM neural network.....	15
Fig. 1.9: Examples of images from the MNIST database .....	16
Fig. 1.10: Thread hierarchy in CUDA.....	17
Fig. 2.1: Waterfall model of software development .....	18
Fig. 2.2: Window for loading the database.....	20
Fig. 2.3: The main window of application.....	21
Fig. 2.4: Tab "Count of Clusters" .....	22
Fig. 2.5: Tab "Clusters".....	23
Fig. 2.6: Tab "Percentage of errors".....	23
Fig. 2.7: Recognition using method 1" .....	24
Fig. 2.8: Recognition using method 2" .....	25
Fig. 2.9: Recognition using Kohonen Neural Network" .....	26
Fig. 2.10: Sequence diagram.....	26
Fig. 2.11: Class for Automatic Clusterization .....	28
Fig. 2.12: Class for Kohonen Neural Network.....	32
Fig. 2.13: The examples of images included in the intersection of clusters .....	35
Fig. 2.14: The percentage of numbers from the test sample in its cluster .....	35
Fig. 2.15: Examples of the image of number 0 from a test sample that did not fall into its cluster .....	36
Fig. 2.16: The percentage of numbers from the test sample in its cluster .....	39
Fig. 2.17: Class for Hopfield Neural Network.....	41
Fig. 2.18: Examples of incorrect recognized images when using method 1 .....	43
Fig. 2.19: Examples of correct recognized images when using method 1 .....	44
Fig. 2.20: Examples of incorrect recognized images when using method 2 .....	46
Fig. 2.21: Examples of correct recognized images when using method 2.....	47
Fig. 2.22: Function that finds the nearest weight (cluster) to the specified vector.....	48
Fig. 2.23: Sequential learning algorithm for the Kohonen neural network .....	49
Fig. 2.24: Distance function that calculates the distance between vectors on a graphics device.....	49
Fig. 2.25: A core that represents a parallel implementation of network learning .....	50

Fig. 2.26: Execution time of training of the Kohonen neural network on the CPU and on the GPU .....	50
Fig. 2.27: Execution time of training of the Kohonen neural network on the CPU and on the OpenMP .....	51
Fig. 2.28: Execution time of the Kohonen neural network training on the CPU for all digits .....	52
Fig. 2.29: Effective CPU utilization histogram .....	52
Fig. 2.30: Execution time of the Kohonen neural network training for all digits using OpenMP .....	52
Fig. 2.31: Effective CPU utilization histogram .....	53
Fig. 2.32: The execution time of the Kohonen neural network training on the GPU and using OpenMP for each digit .....	53
Fig. 3.1: Examples of images from the Fashion MNIST database .....	54
Fig. 3.2: Percentage of images from the test sample that are included in your cluster .....	56
Fig. 3.3: Percentage of image recognition from the test sample by the Hopfield neural network method 1 .....	57
Fig. 3.4: Percentage of image recognition from the test sample by the Hopfield neural network method 2 .....	58

# List of tables

Table 1.1: Classification of neural networks.....	6
Table 2.1: The number of images of each digit in the training and test samples in the MNIST database .....	19
Table 2.2: Description of tabs in TabControl.....	22
Table 2.3: The dependence of the optimal number of clusters on a fixed percentage of the maximum distance for each digit for a sample of 1000 vectors .....	30
Table 2.4: The optimal number of clusters for each digit with a sample of 1000 vectors .....	30
Table 2.5: The dependence of the optimal number of clusters on a fixed percentage of the maximum distance for each digit for a full sample.....	31
Table 2.6: The optimal number of clusters for each digit with a full sample.....	31
Table 2.7: Cluster centers for each digit when selecting 1000 images and dividing by 3 clusters .....	33
Table 2.8: Intersections of clusters when selecting 1000 images and dividing into 3 clusters .....	34
Table 2.9: Examples of clusters with a full sample and an optimal number of clusters.....	36
Table 2.10: Intersections of clusters .....	38
Table 2.11: Examples of images that are included in cluster intersections.....	39
Table 2.12: Summary results of recognition by the Kohonen neural network.....	40
Table 2.13: F-measure .....	40
Table 2.14: Results of recognition by the Hopfield neural network using method 1.....	42
Table 2.15: Summary results of Hopfield neural network recognition using method 1.....	43
Table 2.16: F-measure for the Hopfield neural network using 1 method .....	43
Table 2.17: Results of recognition by the Hopfield neural network using method 2.....	45
Table 2.18: Summary results of Hopfield neural network recognition using method 2.....	46
Table 2.19: F-measure for a Hopfield neural network using 2 method .....	46
Table 2.20: GPU acceleration.....	50
Table 2.21: Computer characteristics .....	51
Table 2.22: OpenMP acceleration .....	52
Table 3.1: The optimal number of clusters for each image with a full sample .....	54
Table 3.2: Examples of cluster centers.....	55
Table 3.3: Summary results of recognition by the Kohonen neural network.....	56
Table 3.4: F-measure .....	56
Table 3.5: Summary results of recognition by the Hopfield neural network .....	57
Table 3.6: F-measure .....	57
Table 3.7: Summary results of recognition by the Hopfield neural network.....	58
Table 3.8: F-measure .....	59

Table 3.9: Examples.....	59
--------------------------	----

# Introduction

Artificial neural networks are used as a method of deep learning. It is one of the many subsections of artificial intelligence. Artificial neural networks were first proposed about 70 years ago as an attempt to simulate the human brain function. Due to advances in hardware development, humanity has been able to build networks that can be trained on a huge set of data in order to achieve breakthroughs in machine intelligence. These discoveries allowed machines to match and exceed the capabilities of humans in performing certain tasks. One of these tasks is object recognition.

In the modern world, the problem of pattern recognition by artificial intelligence is widely used in many industries [1, 2]. Handwriting recognition essentially facilitates working at the computer. In addition to handwritten digits recognition tasks, neural networks are also applied for recognition of other objects, for example, characters [3], facial emotions [4] and much more.

In the present work, we considered the problem of recognizing the patterns of handwritten digits contained in the MNIST database [5]. Recognition was performed using a Hopfield neural network [6]. Due to the fact that the Hopfield neural network has a limited number of objects to "remember", Kohonen neural network was applied as the first stage [7-9]. The results of the first stage were cluster centers. It was these centers that served as the sample for memorization for Hopfield neural network. It is also necessary to compare two methods that serve to determine the similarity of two images.

Today, there are many ways to recognize handwritten digits using neural networks. For example, convolutional neural networks [10]. This type of neural network is the most popular for pattern recognition and shows significant accuracy as a result.

The problem of handwritten digits recognizing lies on the fact that all people have different handwritings, so no images are exactly the same. This issue is being resolved by data clustering methods [14,15]. Kohonen networks are also widely used for recognizing handwritten images and data pre-processing [12,13].

Furthermore, this work raises the question of identifying the optimal number of clusters separately for each digit. Many algorithms have been studied, one of which is presented in a given work [10].

In this paper, the learning algorithm of Kohonen neural network was parallelized using CUDA technology. The results were compared with work on a single processor as well as with the results of parallelizing using OpenMP.

The novelty of the work consists in the following aspects:

1. We propose a hierarchical pattern recognition model that consists of two neural networks: Kohonen and Hopfield. Kohonen neural network pre-processes data and outputs a sample for the Hopfield neural network as a result. Hopfield neural network "remembers"



- images and recognizes any particular image.
2. Hopfield neural network offers 2 methods for comparing two images: pixel-by-pixel comparison and comparison involving the f-metric.
  3. The analysis of the MNIST database employing the automatic clustering reveals the optimal number of clusters the base needs to be divided in.
  4. It is proposed to parallelize the learning algorithm for Kohonen neural network using the CUDA technology.

## Thesis content

The goal of this work is to develop and create a software module that will build a hierarchical neural network for pattern recognition.

The task consists of the following stages:

1. Writing an automatic clustering algorithm to identify the optimal number of clusters separately for each digit of the MNIST database.
2. Writing a Kohonen neural network separately for each digit.
3. Writing a Hopfield neural network for pattern recognition, where clusters that were obtained as a result of applying a Kohonen neural network serve as objects to remember. In Hopfield neural network, 2 methods are compared to identify two similar images.

The work consists of three main chapters. The first chapter describes the biological description of neural networks and their artificial analogues. It describes the methods of neural networks training, Kohonen neural network, which is used as one of the stages in this work, the methods of standard and automatic clustering, Hopfield neural network. The chapter overviews popular image recognition technologies, and presents a brief description of the MNIST database, which was tested. There is also an item that explains the methods of parallelization of neural networks and CUDA technology. The second Chapter contains a description of the software, results and analysis of the work as well as testing the training of Kohonen neural network on GPUs using CUDA technology. The training of Kohonen neural network was analyzed using Intel Parallel Studio. The third Chapter demonstrates testing of the system and algorithms. The third chapter demonstrates testing of the system and algorithms.

The program was written in the object-oriented C# language. The system was tested on the basis of handwritten MNIST numbers, as well as on the basis of Fashion MNIST, which contains images of different types of clothing. This demonstrates the versatility of the algorithms. Parallelization was performed in C++. All programs were written in Visual Studio 2019.

This thesis consists of the introduction, theoretical parts, practice part, practical and conclusion.

# Chapter 1

## Neural networks and MNIST database

### 1.1 Artificial neuron and artificial neural networks

#### Biology neuron

In 1888, Dr. R. Kayal showed that brain tissue consists of a huge number of similar nodes that are connected to each other. These nodes are called neurons. A biological neuron is a cell that stores and transmits information using electrical and electrochemical impulses. A single neuron can transmit information to another neuron as well as to another type of cell.

All neurons have a similar structure. Each neuron has:

- Body (Soma) – vital intracellular processes occur in it. The neuron body contains the nucleus that carries the neuron's genetic material.
- From the central part of the neuron emerge dendrites-a process that receives impulses from other cells. Depending on the cell type, a single neuron can have up to 200 dendrites.
- An axon is a nerve fiber that serves as an element through which a nerve impulse passes. The aim of an axon is to transmit a signal from a neuron to other neurons.

The place where axon and dendrites connect is called a synapse. The synapse is a synaptic gap of about 200 nm, white matter and synaptic protrusions-a nerve signal is sent through them. The mechanism that underlies signal transmission within neurons is based on the potential difference that exists between the inner and outer part of the cell. This membrane potential is created by an uneven distribution of electrically charged particles or ions: sodium ( $\text{Na}^+$ ), potassium ( $\text{K}^+$ ), chloride ( $\text{Cl}^-$ ), and calcium ( $\text{Ca}^{2+}$ ). At rest, the sodium ions are evenly distributed, so the neuron has a negative charge. In order to transmit a nerve impulse, you need a stimulus – special chemicals- neurotransmitters. When the mediator gets the dendrite, the charge of the neuron changes from negative to positive. The dendrite is attracted to a negatively charged axon, and along the axon reaches the synapse-the charge reaches another cell. Figure 1.1 shows a biological neuron.

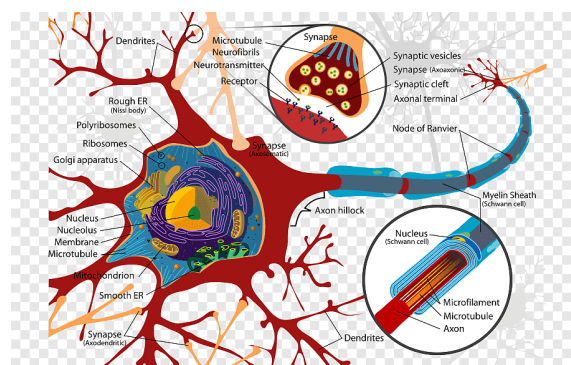


Fig. 1.1: Neuron

**Artificial neuron**

The first work that laid the foundation for creating artificial models of neurons and neural networks was an article by V. McCulloch and V. Pitts "Logical calculus of ideas related to neural activity". McCulloch and Pitts proposed an information processing system in the form of a network that consists of n simple computing neurons. Each calculator consists of inputs and outputs that take the value 0 or 1.

The state of a single neuron is calculated as  $\sum w_{ij}n_j$  – a weighted linear combination of their outputs. Then the threshold function is applied to the amount. The threshold function that was considered by McCulloch and Pitts has the form:

$$Y = f(\sum wn) = \begin{cases} 1, \sum wn > 0 \\ 0, \sum wn \leq 0 \end{cases} \quad (1)$$

Such networks can perform calculations like a Turing machine. But the only unresolved problem is how to initially initiate the  $w_{ij}$  weights.

In 1962, Rosenblatt proposed the following model of a neural network – the perceptron-Fig. 1.2.

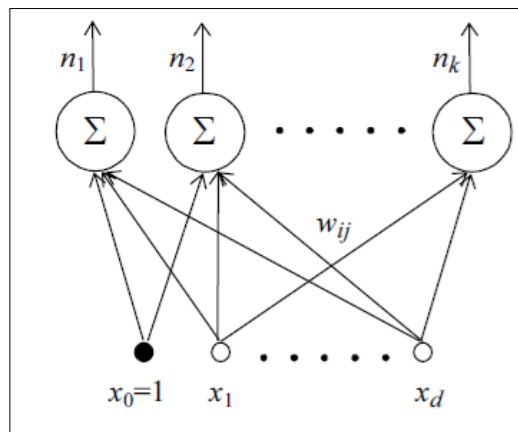


Fig. 1.2: Rosenblatt's Perceptron

The Rosenblatt perceptron consists of k neurons, d inputs and d outputs, and a single layer of configurable  $w_{ij}$  weights. Each perceptron neuron has the following structure – Fig. 1.3.

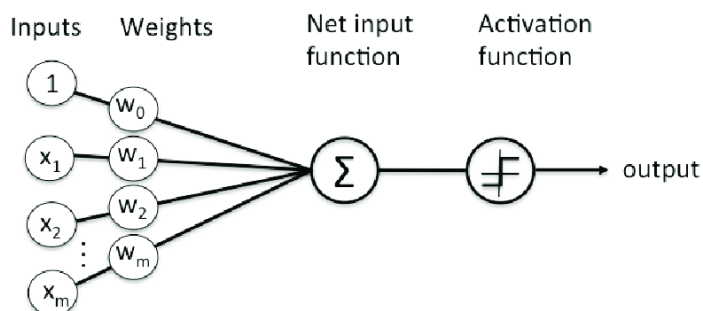


Fig. 1.3: Artificial neuron

Each neuron calculates a weighted sum of inputs:

$$y_j(x) = \sum_{i=0}^d x_i w_{ij} \quad (1.2)$$

The output of a neuron is calculated using a nonlinear threshold function of the form:

$$g(a) = \begin{cases} 1, \sum wn \geq 0 \\ -1, \sum wn < 0 \end{cases} \quad (1.3)$$

Rosenblatt suggested to use a perceptron to solve the classification problems by Rosenblatt. Let us analyze a simple case as an example. Let there be a set of vectors of the form  $x=(x_1, x_2)$ , where each one belongs to one of two different classes  $C_1, C_2$ . This set will be a sample for training. The challenge lies in determination of belonging to the class for any vectors. Drawing on equation (1.2), we can conclude that the neuron in the case of a hyperplane builds a straight line on the plane. Therefore, the vectors coming to the input will fall into one or another half-plane. The threshold function defines the half-plane that accepts the input vector. The type of straight line is determined by the  $w_{ij}$  scales that are configured during training.

For the correct operation of the neural network – perceptron – it is necessary to minimize the error function:

$$E(w) = -\sum_{x^n \in M} W^T(xt), \quad (1.4)$$

where  $M$  is a set of incorrectly classified data for  $w_{ij}$  weights,  $WTx$  is the matrix form of the record (2),  $\{t_n\}$  is the set of desired neuron outputs. Formula (1.4) is called the perceptron criterion.

The following is a perceptron learning algorithm based on a training sample:

1. Initially,  $w_{ij}$  weights are initiated randomly.
2. A vector from the training sample is provided to the input of the neural network.
3. At the output of the following situations are possible:
  - if the neural network classifies the vector correctly, we do nothing;
  - If the perceptron classifies the vector incorrectly and the vector must belong to class  $C_1$ , that is, the output of the neuron after applying the threshold function is  $+ 1$ , then the vector is added to the weight;
  - If the perceptron classifies the vector incorrectly and the vector must belong to class  $C_2$ , i.e. the output of the neuron after applying the threshold function is  $- 1$ , then the vector is subtracted from the weight.

The last two sub-items of change in weights can be represented as:

$$W^{i+1} = W^i + \eta(wt), \quad (1.5)$$

where  $\eta > 0$  is the coefficient that sets the learning speed, and  $W^{i+1}$  is the matrix of new weight values. When training a neural network, the criteria will decrease. Since the learning rate coefficient is greater than zero and  $(xt)^2 > 0$ , then:

$$E^{i+1} = -W^{i+1}(xt) = -W^i(xt) - \eta(xt)(xt) < W^i(w) = E^i(w) \quad (1.6)$$

In 1973, it was proved that for any linearly separated input data, the perceptron learning algorithm finds a solution in a finite number of steps. But by increasing the number of layers, you can solve problems not only for linearly separated input data, but also for problems of any complexity. A perceptron with several layers is called a multi-layer perceptron.

## 1.2 Classification and training of neural networks

Currently, all neural networks can be divided into two groups: feedforward networks and feedback networks. In feedforward neural networks, the signal moves in one direction: from the network entrance to its exit. In back propagation networks the signals can move back to the inputs of the network, such structures allow us to solve more complex problems. But the outputs of such networks are unstable. The table below shows the known types of neural networks of the first and second types.

Feedforward networks	Feedback networks
Perceptron	Counter-propagation
Multi-layer perceptron	SOM, Kohonen neural networks
RBF-networks	Associative memory, Hopfield networks
Cascade correlation networks	An Elman Network
Adalin	ART-networks
	Stochastic networks (Boltzmann machine)
	Time Delay-networks

Table 1.1: Classification of neural networks

Setting up artificial neural networks is divided into 3 categories: training "with a teacher", training "without a teacher" and training by the method of criticism.

In the "teacher training" approach, for each object  $x \in \{x_n\}$  from the training sample, the result  $t \in \{t_n\}$  that we would like to get is known in advance. The main disadvantage of this method of training is that there are not always enough examples with a pre-known result.

In the "learning without a teacher" approach, the desired result is not known in advance for the training sample. In this case, the weights are adjusted at the discretion of the neural network. The neural network finds patterns between data and classifies them into groups where objects are

similar to each other. An example of such a network is Kohonen neural network.

With the critical learning approach, it is possible to evaluate how well the neural network works and then specify the desired direction of training. This approach is an intermediate step between "with a teacher" and "without a teacher" training.

### 1.3 The Kohonen neural network. SOM

In this graduation work, there is a database of handwritten MNIST numbers. The first step implies building of the main clusters separately for each digit (0,1,2, etc). Kohonen neural network was used for this purpose.

As was said earlier, Kohonen neural networks is a type of artificial neural networks that are trained "without a teacher". Learning "without a teacher" is close to the real work of the human brain. The main part of the information that comes to the visual, auditory, tactile and other receptors comes from outside, and then inside the nervous system a certain classification and organization are made. The Kohonen algorithm provides for weights setting based on their values in the previous iteration.

The architecture of the Kohonen neural network (Fig. 1.4) was proposed in 1987 by Teuvo Kohonen.

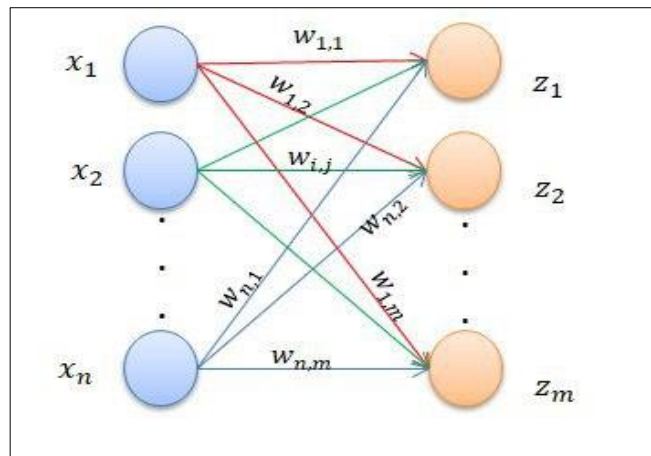


Fig. 1.4: Architecture of the Kohonen neural network

The Kohonen neural network consists of a single layer of customizable scales and functions in the spirit of the strategy: "the winner takes it all" – only one neuron fires, the remaining outputs of the layer are suppressed.

The outputs  $z_j$  are calculated by the following formula:

$$z_j = \sum_i w_{ij}x_i, \quad \exists k, z_k = 1, z_{j \neq k} = 0, \quad (1.7)$$

where  $z_j$  are the outputs of the Kohonen network. The Kohonen network combines input vectors into a group of similar ones – clusters. In this case, the weights of the neural network change so that vectors belonging to the same cluster activate the same output neuron.

For the training algorithm of the Kohonen neural network, it is necessary that the input

vectors are normalized – reduced to a single vector in space. Normalization takes place according to the following formula:

$$x'_i = \frac{x_i}{\sqrt{\sum_i x_i^2}}, \quad (8)$$

After the input vectors are normalized, the following algorithm is executed:

1. Weights  $w_{ij}$  are filled in with initial values. Usually, in neural network training algorithms, the initial values of weights are filled in with random numbers. In the Kohonen neural network, the weights of output neurons are evenly distributed on the surface of the hypersphere. The problem is that most often the input vectors  $x_i$  are distributed unevenly; the weight vectors of most of the neurons are removed from the input ones, so they will not participate in the learning algorithm, and the remaining vectors are not enough to divide into classes. To solve this problem, it is proposed to initiate weights by placing most of them in a cluster of input vectors. We suggest to use the convex combination method: all weights are made equal  $w_{ij} = \frac{1}{\sqrt{N}}$ , where  $N$  – dimension of the input vector. The input vectors change according to the formula (1.9), and the coefficient  $\alpha$  tends to 0, so all input vectors have the form:  $x_i \approx \frac{1}{\sqrt{N}}$ . In the process of learning the coefficient increases to 1.

$$x_i = \alpha x_i + \frac{1-\alpha}{\sqrt{N}}. \quad (1.9)$$

2. There are two options for applying vector  $x_i$  to the input. In the first case, the input is a normalized vector  $x_i$ , then the scalar product with the weight vector of each neuron  $w_j$  is calculated using the formula:

$$x = |x||w|\cos(xw). \quad (1.10)$$

The vector that gives the maximum scalar product is selected. In the second case, the input is a normalized vector  $x_i$ , then the distance between the vectors  $w_j$  and  $x$  is calculated using the formula (1.11). The weight vector with the smallest distance to it is selected.

$$D_j = \left( \sum_{i=1}^n (x_i^p - w_{ij})^2 \right)^{1/2} .. \quad (1.11)$$

3. The neuron selected in the previous step is configured according to the formula:

$$w^{t+1} = w^t + \theta(x - w^t), \quad (1.12)$$

where  $w^{(t+1)}$  is the new value of the neuron weight,  $\theta$  is the learning speed. Setting up weights reduces to the task of minimizing the difference between the input vector and the weight vector for the selected neuron.

4. Items 2 and 3 are repeated until the network outputs are less than the specified

accuracy.

## 1.4 Clustering methods

In present work, the Kohonen neural network is proposed to solve the clustering problem. But there are many other algorithms for resolving this issue. Different clustering algorithms can be successful in different situations. Creating an algorithm that will accurately determine clusters for all situations is quite difficult and hardly solvable task.

Graph algorithms, such as connected components and the shortest open path, provide more visibility. The disadvantage of these algorithms is that they work slowly and often are sensitive to various outliers that are formed by atypical objects.

The FOREL algorithm is as visual as graph algorithms, but more stable. The algorithm builds a two-level cluster structure, that is very convenient to interpret. It is successful in practical applications.

The EM algorithm is one of the most popular algorithms for solving clustering problems. It allows you to determine the optimal number of clusters to split the selection into. The algorithm is able to identify outliers that are formed by atypical objects. But the EM algorithm is sensitive to the initial approximation.

The k-means algorithm is a simpler version of the EM algorithm and is also very popular due to its ease of implementation. The advantage is that the algorithm converges faster than the EM algorithm. But the algorithm is even more sensitive to the initial approximation. Therefore, it is mostly poorly clustered.

The hierarchical agglomerative clustering algorithm allows you to identify a detailed cluster structure in the form of a dendrogram.

## 1.5 Automatic clustering methods

In this work, a training sample from the MNIST database of handwritten digits contains 60,000 images. There are many ways to write a particular number. In classical clustering methods, such as the k-means method and the nearest neighbor method, the number of clusters to split the sample into is known in advance. The number of clusters for each digit will be different. To do this, you need to use automatic clustering, i.e. find out how many clusters you need to split the selection into.

There are several algorithms. In this work, we apply the following algorithm:

1. Calculate the squares of distances between all vectors in the sample, using the Euclidean metric:

$$d^2(x^i, x^j) = \|x^i - x^j\|^2 = \sum_{l=1}^n (x_l^i - x_l^j)^2, \quad (1.13)$$



From the obtained values, we form a matrix.

2. Determine the maximum element of the matrix (the maximum distance among all vectors):  $\max d^2(x^i, x^j)$ .
3. Determine the allowable distance between the vectors in the same cluster. The permissible distance is set by a fixed percentage of the maximum matrix element (maximum distance between vectors).
4. Select an arbitrary  $i$ -th column of the matrix (vector  $x^i$ ). Mark all elements of the column, whose values are less than permissible, as elements of the same cluster (rows with number  $j$ ). We exclude the  $i$ -th column and, as well as all the  $j$ -th columns.
5. If the matrix is not empty yet, then go to the step 4.

Also, there is another algorithm:

1. The distances from each vector to all the others are calculated using the formula (1.13).
2. Determination of the maximum element of the matrix (the maximum distance among all vectors):  $\max d^2(x^i, x^j)$ .
3. Normalization of distance according to the formula:

$$d^2(x^i, x^j) = \frac{d^2(x^i, x^j)}{\max(d^2(x^i, x^j))} \quad (1.14)$$

4. Distances are sorted in ascending order.
5. Clustering vectors: if the next distance value in one column is no more than the previous value  $n$  times, then the vectors are in the same cluster. The parameter  $n$  is fixed.

Both of the above algorithms give approximately the same results, but the processing time of the second algorithm is much higher for a large amount of data. Therefore, in this work, the first algorithm is used to determine the optimal number of clusters.

## 1.6 Recurrent neural networks

Among the many networks, there are artificial neural networks where weights are calculated only once before the neural network starts functioning. In fact, the network stores information for training before test data is submitted for input.

If you look at the architecture of such networks, there are feedbacks in recurrent networks, in contrast to feedforward networks: the signal from the output of the neuron can go back to the input. Feedbacks allow neural network outputs to accept arbitrary states at different times with the same inputs without necessary stabilizing on the same state.

The feedback neuron is shown in Fig. 1.5.

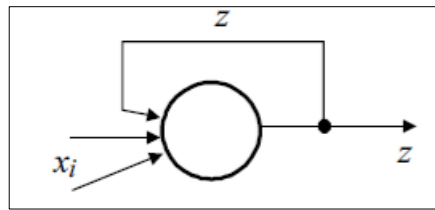


Fig. 1.5: The feedback neuron

The work of a feedback neuron is as follows. First, the values  $x_i$  are fed to the input of the neuron and the output of the neuron  $z$  is calculated. Then the output value is again fed to the input of the neuron along with other values and a new value is calculated. This process ends when the output  $z$  value changes insignificantly from iteration to iteration. From here, we can divide recurrent networks into two classes: stable and unstable. Stable networks are recurrent neural networks for which it is possible to get output values that have stabilized to a single value. Unstable ones are those whose output values are not stable.

Among the networks that operate under this arrangement, the most popular is a Hopfield neural network. Hopfield neural network consists of a single layer of configurable  $w_{ij}$  weights. Every neuron in the network is connected to all neurons: all neurons return their outputs to their inputs. There is also one input synapse through which the signal is entered. Hopfield neural network operates with the values +1 and -1. The Hopfield neural network architecture is shown in Fig. 1.6.

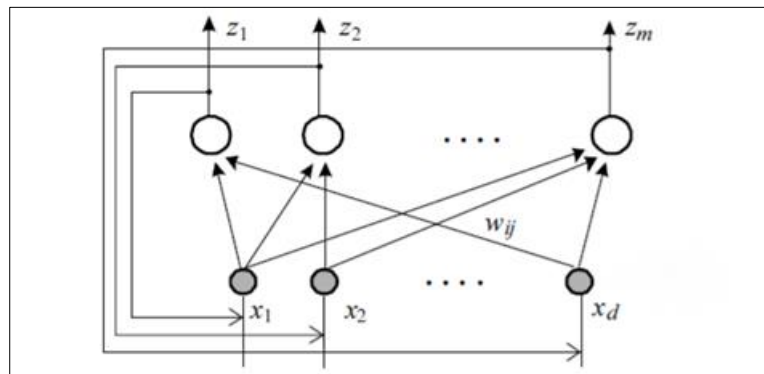


Fig. 1.6: Architecture of Hopfield neural network

The problem that is solved using this network is formulated as follows. Let us give a set of binary signals: images, audio digitizations, and so on. Hopfield network must select an imperfect signal from the input ("remember») the corresponding sample, if there is one in memory, or give an exception that the input signal does not match the existing samples.

Input signals are sent once to Hopfield neural network. Each signal from the training sample is described by the vector  $X=\{x_1, x_2, \dots, x_m\}$ , where  $m$  is the dimension of the vector  $x$ . This is the set that the neural network "remembers". The input signal that needs to be recognized or "remembered" is denoted by  $Y=\{y_1, y_2, \dots, y_m\}$ , where  $m$  is the dimension of the  $y$  vector. Let  $X_k$  be a vector that describes the  $k$ -th signal. If the Hopfield network recognizes any sample, then  $X_k=Y$ , otherwise the output vector will not match any vector from memory.

Before Hopfield neural network starts functioning the  $w_{ij}$  weights are calculated using the formula:

$$w_{ij} = \begin{cases} \sum_{k=0}^{m-1} x_i^k x_j^k, & i \neq j \\ 0, & i = j \end{cases} \quad (1.15)$$

Then Hopfield neural network starts working.

1. A Y signal is sent to the input of the neural network, that must be recognized.
2. Calculation of the new state of neurons:

$$S_j(p+1) = \sum_{i=0}^{n-1} w_{ij} y_i, \quad (1.16)$$

where p is an iteration number.

3. The new state of axons is also calculated:

$$y_j(p+1) = f[S_j(p+1)], \quad (1.17)$$

where f is a threshold function of the form:

$$f(S) = \begin{cases} 1, & S_j > T_j \\ -1, & S_j < T_j \\ \text{not changes,} & S_j = T_j \end{cases}, \quad (1.18)$$

where  $T_j$  is the threshold of the neuron.

4. Check whether the output of neurons has changed over the last iteration. If it has, go to the step 2, otherwise the outputs have stabilized. If there is no vector exactly matching the vectors in the memory of the neural network, the answer is the sample that best matches the input.

A Hopfield neural network is stable if the weight matrix  $W$  is symmetric, that is,  $w_{ij}=w_{ji}$  and the values on the main diagonal are 0. This can be proved as follows. Consider a neural network as a dynamic system that has an energy state. The energy of the system is denoted by  $E$ , then if the energy of  $E$  decreases or does not change when the state make any changes, so the system will be stable by definition. Based on the symmetry condition of the system and the condition that the values on the main diagonal are 0, we can describe the energy as follows:

$$E = -\frac{1}{2} \sum \sum w_{ij} z_j z_i - \sum x_j z_j + \sum T_j z_j, \quad (1.19)$$

where  $w_{ij}$  is the weights of the neural network,  $z_j$  is the output value of neuron j,  $x_j$  is the j-th element of the input vector, and  $T_j$  is the threshold of neuron j. The change in energy  $E_j$  due to a change in the state of  $z_j$  is calculated using the formula:

$$\Delta E_j = -[\sum_{i \neq j} (w_{ji} z_i) + x_j - T_j] \Delta z_j = -(S_j - T_j) \Delta z_j. \quad (1.20)$$

Then, based on the formula (1.20), there are 3 cases:

1.  $S_j > T_j$ . Then  $(S_j - T_j) > 0$ , therefore substituting this in (1.18),  $\Delta z_j$  is either greater than zero, or will not change. Then  $\Delta E_j$  decreases or stabilizes.
2.  $S_j < T_j$ . Then  $(S_j - T_j) < 0$ , therefore substituting this in (1.18),  $\Delta z_j$  is either less than zero,

or will not change. Then  $\Delta E_j$  decreases or stabilizes.

3.  $S_j=T_j$ . Then  $\Delta E_j=0$ , i.e. stabilizes.

Thus, any change in state reduces the energy, and therefore the neural network, therefore the Hopfield network is stable.

Hopfield neural network forms a simplified model of associative memory. Human memory is associative, meaning that the brain receives information and receives a lot of memories in response. For example, if a person hears a person's name, the brain will give out information about the appearance of this person, the emotions that this person causes, and many other memories. Associative memory has a very complex structure. The algorithm for working with associative memory has the following form:

1. All  $x_j$  images that need to be remembered are represented by vectors of length  $N$  and take the values  $+1$  or  $-1$ .
2. Weights of the neural network are initialized using the formula:

$$w_{ij} = \sum_{d=0}^M x_d^i x_d^j \quad (1.21)$$

3. After the images are stored in memory, the process of restoring memories begins. Due to feedback, the distorted image can be restored. But if the image was badly distorted, the result may be incorrect.

The following question occurs: what is the capacity of the network, that is, how many images can the Hopfield neural network remember. Experimentally, it has been shown that the Hopfield neural network, which consists of  $N$  neurons, can only remember fifteen percent of  $n$ . Thus, associative memory, which is implemented using the artificial Hopfield neural network is associative.

## 1.7 Overview of neural networks for pattern recognition

The task of this work is to recognize images of handwritten numbers from the MNIST database using the Kohonen and Hopfield neural network. The problem of pattern recognition by neural networks is one of the most popular today. Therefore, there are many neural network architectures for solving this problem.

1. Multilayer perceptron;
2. Convolutional neural network;
3. Recursive neural network;
4. Recurrent neural network;
5. Network of long-term short-term memory (LSTM);
6. Sequence-to-sequence model;
7. etc.

One of the most popular and frequent neural network architectures is the multilayer perceptron. But one of the biggest disadvantages of this network is its redundancy. For example, if

the input is an image in the form of a 34x34 matrix, the neural network will have 1156 inputs. This indicates a large amount of computing power that is expended for this algorithm. A study was conducted to solve this problem. It was found that there are two types of cells in the visual cortex of the brain: simple and complex. Simple cells respond to the image of straight lines, and complex cells to forward movement in one direction. Based on this research, an algorithm for convolutional artificial neural networks was constructed.

In convolutional neural networks, each input neuron gets not the entire image, but only a part of it. Due to this, you can save the topology of the image from layer to layer and use in the processing of multiple neural networks. For the many connections in a neural network a small set of weights can be used, these sets are called nuclei. Shared weights increase the ability to find invariants in images and filter out the noise.

Thus, convolutional neural networks have a number of advantages: a reduction in the number of trained objects, and therefore a high learning speed compared to a multi-layer perceptron; the ability to parallelize; the ability to shift the position of the object. Convolutional neural networks are the best algorithms for facial recognition in terms of accuracy and speed.

The brain has a feature: receiving any information, at the output of the brain will give a lot of other information that is somehow related to the input. Since traditional neural networks cannot mimic this feature, recurrent neural networks have been developed. As mentioned earlier, one of the most popular recurrent neural networks is a Hopfield neural network. Recurrent neural networks are successful in a number of tasks. Great results were achieved by LSTM (Long short-term memory) networks. The LSTM network is a modification of the standard recurrent network that is capable of learning long-term dependencies. Let's look at an example of predicting a word in a sentence. Sometimes, in order to accurately predict information, it is enough to refer to recent information. Standard recurrent networks easily solve this problem, but if a broader context is needed for forecasting, errors occur. This is exactly what LSTM networks do. A standard recurrent network has the form of a chain of repeated modules, where one module consists of a single layer with an activation function. In LSTM networks, the module contains four layers. The Fig. 1.7 and the Fig. 1.8 show the architecture of a regular recurrent network and an LSTM network.

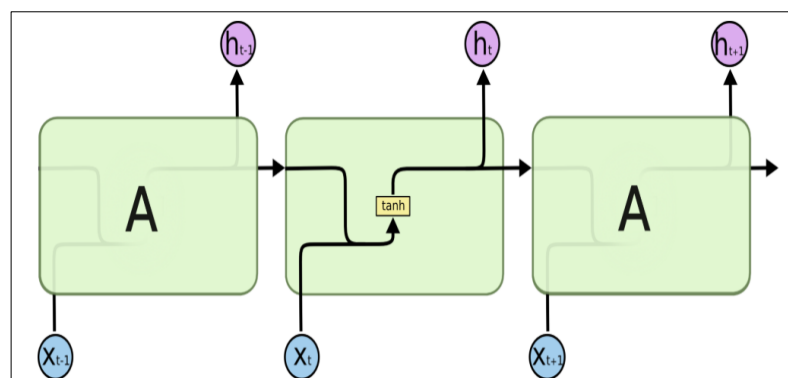


Fig. 1.7: Architecture of a standard recurrent neural network

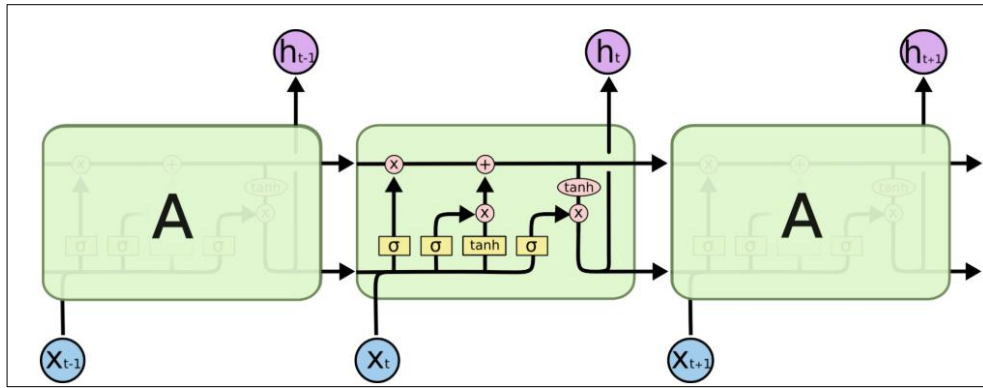


Fig. 1.8: Architecture of a LSTM neural network

LSTM networks are a big achievement in recurrent neural networks. The disadvantage is their complexity, but their ability to mimic the operation of long-term memory is a huge advantage. According to the researchers, the next step in recurrent networks will be to simulate the attention mechanism.

## 1.8 MNIST database

The work was tested on the basis of handwritten numbers in the MNIST database (test sample). The MNIST database is a large data set consisting of samples of handwritten digits. The MNIST database was published by the US National Institute of standards and technology.

The database contains 60,000 samples for the training sample and 10,000 samples for the test sample.

Each image is represented in two ways: as a placemark and as a vector of pixel values. Each image in the database is normalized in size. The size of each image is 28 by 28 pixels. The pixel value varies from 0 to 255: 0 is a black pixel, and 255 is a white pixel.

Many methods have been tested on this database. For example, K-nearest neighbor methods based on MNIST give an error of 5%, convolutional neural networks give an error of less than 1%, multi-layer perceptron an error of about 2-5% depending on the training methods and the number of layers.

MNIST database is perfect for training neural networks for pattern recognition, because in addition to clearly written numbers, the network has large handwriting distortions. Therefore, the network is trained not only on standard images. Examples of images from the MNIST database are shown below.

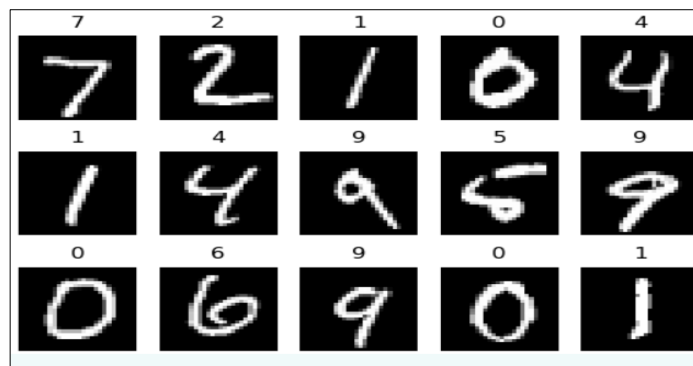


Fig. 1.9: Examples of images from the MNIST database

## 1.9 Using CUDA technology for neural networks

Combining multiple processors to solve a given task is called parallel processing. Parallel processing allows you to speed up the running time of the algorithm. Parallelization is often used during the work with neural networks, because neural networks operate with a large set of data.

There are several levels of parallelization in artificial neural networks:

1. Level phases of training;
2. Level of training sample;
3. The layer level;
4. The level of neurons;
5. Level of weights.

In order to choose at which level to implement parallelization, you need to start from the architecture of the neural network and the task at hand. You can use all levels at the same time.

At the level of learning phase the study of network is made under different primary systems. Using this level of training causes a rapid linear increase in the speed of the phase, that is an essential benefit of applying this level.

At the level of training samples, the network training takes place simultaneously on multiple processors. This is very important, because neural networks often have large training samples.

In multilayer neural network architectures (models with reverse error propagation), vectors follow each other. When using parallelization at the layer level the vectors as well go through the processors. In this case, the most accurate vectors that have passed through all processors are taken as a reference. After the vectors have passed through the processors again, the processors update the weights in the neural network based on the reference values.

When using parallelization at the neuron level, the processing processor is similar to a neuron. Neurons are divided into processors within a single layer. This is one of the most popular methods of parallelization of neural networks.

The weight level is most often used in hardware implementations. Computation in a single neuron (recalculation of the weights, for example) is divided among the processors.

This work uses two neural networks—a Kohonen neural network and a Hopfield neural network. Parallelization was applied for Kohonen neural network. This is due to a large training sample and a large number of clusters to divide the training sample into.

Modern GPUs are very efficient in parallel computing. CUDA technology allows you to write programs using GPUs in a high-level programming language.

CUDA technology only works with graphics devices from NVIDIA. CUDA currently supports the following programming languages: C/C++, Fortran, and Python. In a program written on a standard processor, you can add elements to work on the GPU. The source code must be pre-processed. After that, the standard compiler is called and the code is generated separately for the CPU and separately for the GPU. When computing on a GPU, a large number of parallel processes – threads—run simultaneously. All threads are combined into a grid. A grid is a one-dimensional, two-dimensional, or three-dimensional array of blocks. Each block, in turn, is a one-dimensional, two-dimensional, or three-dimensional array of threads (thread). Blocks have the same dimension.

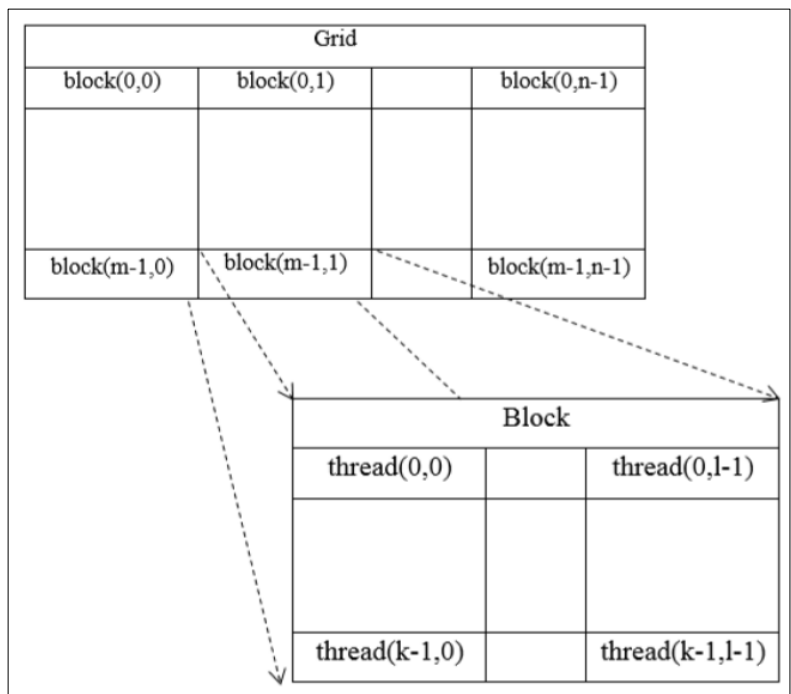


Fig. 1.10: Thread hierarchy in CUDA

The original task is divided into several separate subtasks that are solved independently of each other, and each subtask corresponds to a block of threads. Each subtask is solved only by threads from its own block. Threads can only interact with each other within their own block. They can not interact with threads from other blocks. In order to avoid problems with simultaneous operation, barrier synchronization is used. This type of synchronization does not allow threads to execute commands until all threads perform an action.

Thus, parallelization of neural networks will help you optimize your work over time. And CUDA technology provides very convenient and affordable tools for this.



# Chapter 2

## Description of the software implementation and results

### 2.1 Task statement and program structure

In this graduated work, a cascade model of software development was chosen. This choice is due to the fact that the software is relatively small and all the requirements are precisely formulated. The cascading model assumes that all stages will be completed, and the transition to the next stage is possible only after the previous one has been completed and verified. The structure of the cascade model is shown in Fig. 2.1

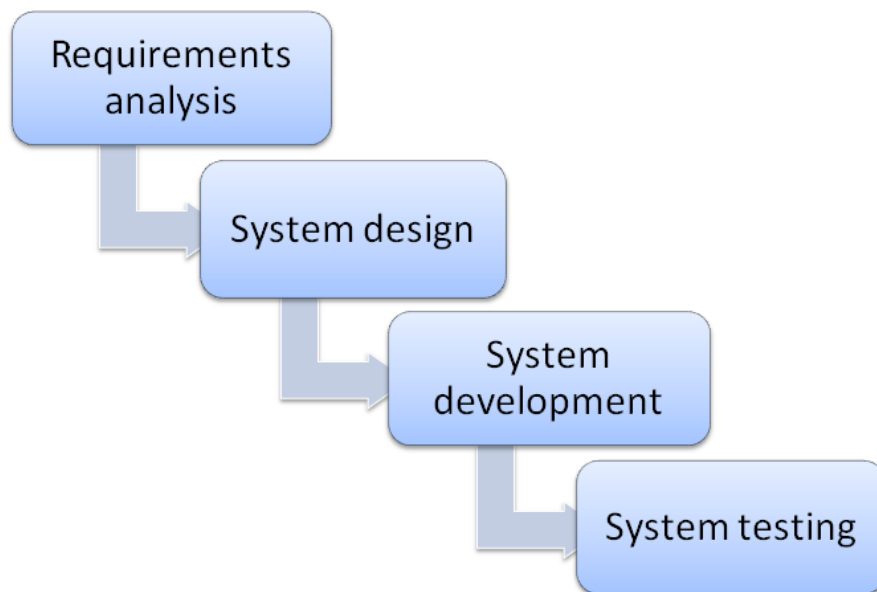


Fig. 2.1: Waterfall model of software development

At the stage of requirement analysis, it is necessary to identify problems and tasks that the software will solve. The system design stage describes the implementation of the system. At the stage of system development, software is directly developed according to the specification obtained at the stage of system design. At the last stage, the system is tested. In this case, the software was tested using a test sample from the MNIST database. Let us list the main points of each stage of the cascade model of software development.

During the requirements analysis stage, the tasks assigned to the developer were identified. In this paper, the task of recognizing handwritten images is set. It was necessary to build a hierarchical neural network that consists of Kohonen neural network and Hopfield neural network. The user must be able to load the database, train both neural networks, and perform cluster analysis using Kohonen neural network.

At the stage of system design, the development environment was selected, as well as the

graphical user interface was designed. The C# programming language was chosen and development was carried out in Windows Forms. The Visual Studio development environment provides a user-friendly development interface and all the necessary tools to create a user-friendly user interface.

At the development stage, all neural networks were developed and an object-oriented programming approach was used.

Testing was conducted, as already mentioned, on a test sample of the MNIST database. Both neural networks were tested. Testing software demonstrates the correctness and accuracy of the algorithm and software in general. The software was also tested on the basis of Fashion MNIST, which demonstrates the versatility of the algorithms. Next, we will discuss the stages of building software in more detail. Logging is performed using the framework NLog.

In this work, we considered the problem of recognizing images of handwritten numbers contained in the MNIST database. Recognition was performed using a Hopfield recurrent neural network. Separately for each digit, we used Hopfield's own neural network that was trained on a sample containing only an image of this individual digit. The table below shows the number of images of each digit in the training and test sample.

Digit	Count in the training sample	Count in the test sample
0	5923	980
1	6742	1135
2	5958	1032
3	6131	1010
4	5842	982
5	5421	892
6	5918	958
7	6265	1028
8	5851	974
9	5949	1009

Table 2.1: The number of images of each digit in the training and test samples in the MNIST database

As you can see from Table 2.1, for each digit in the MNIST database, there are about 6 thousand objects from the training sample. As mentioned in the previous chapter, a Hopfield neural network has a limited number of objects to remember. Therefore, the network will not be able to remember about 6 thousand images. The network capacity is about 15 percent of  $N$ , where  $N$  is the image dimension. The size of each image in the database is 28 by 28 pixels, so  $N=28*28=784$  pixels. Thus, a Hopfield neural network can remember about 100 images. In this paper, to get more accurate information, 50 images were taken for the network capacity.

Due to the fact that a Hopfield neural network has a memory limit, it is necessary to build a small sample. To do this, it was decided to cluster the training sample for each digit using a Kohonen neural network, and cluster centers will be used as the new training sample.

You will also be asked how many clusters each selection should be divided into. To solve this problem, we used automatic clustering, the algorithm of which was described in the first chapter.

When a Hopfield neural network is running, it is not always possible to bring a distorted

image to any image in the network memory. In this case, the result is the image that is the most similar to any image from memory. In this paper, 2 methods were proposed and a comparison analysis was performed. After the 10 neural networks for each image finish their work, the results of these networks-the percentage of how similar the image is to what is being fed to the input-are sorted in descending order – from the most similar to the most dissimilar. The user is shown the first 3 images.

The program works as follows.

1. The user sees a window for loading the database: he can choose the MNIST database that was used for testing, or choose a database from the file system. Before doing this, the user must fill in the information about the database (how much data is for training, how much data is for testing, image format). The graphical interface is configured for the MNIST database, but the recognition algorithms are suitable for other databases, and the data must be pre-processed.

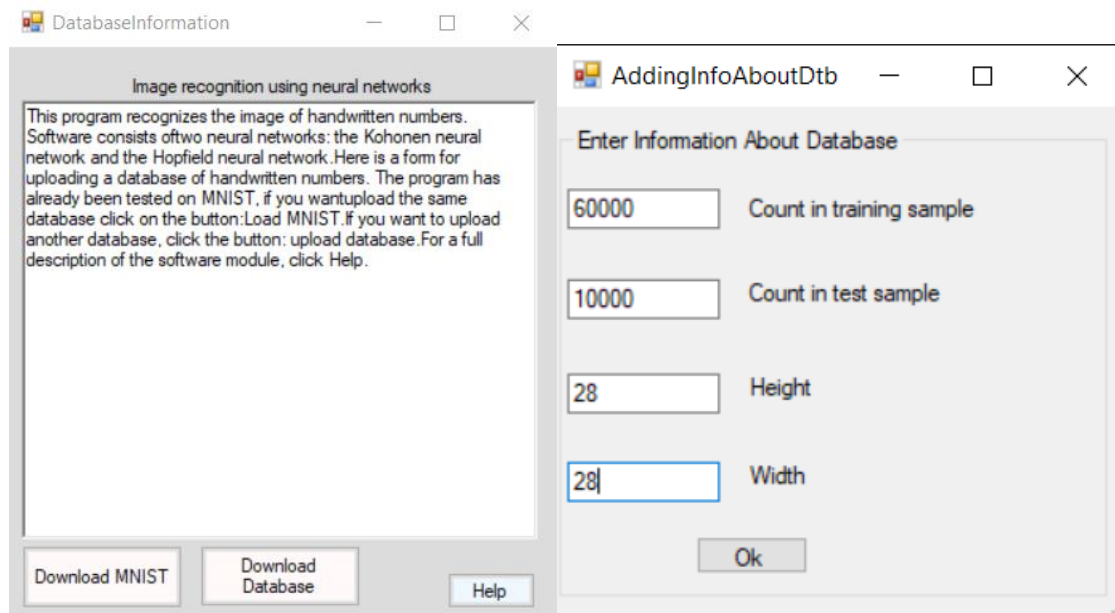


Fig. 2.2: Window for loading the database

2. After the user successfully enters the database, a window opens to operate the program. The window contains the following elements:

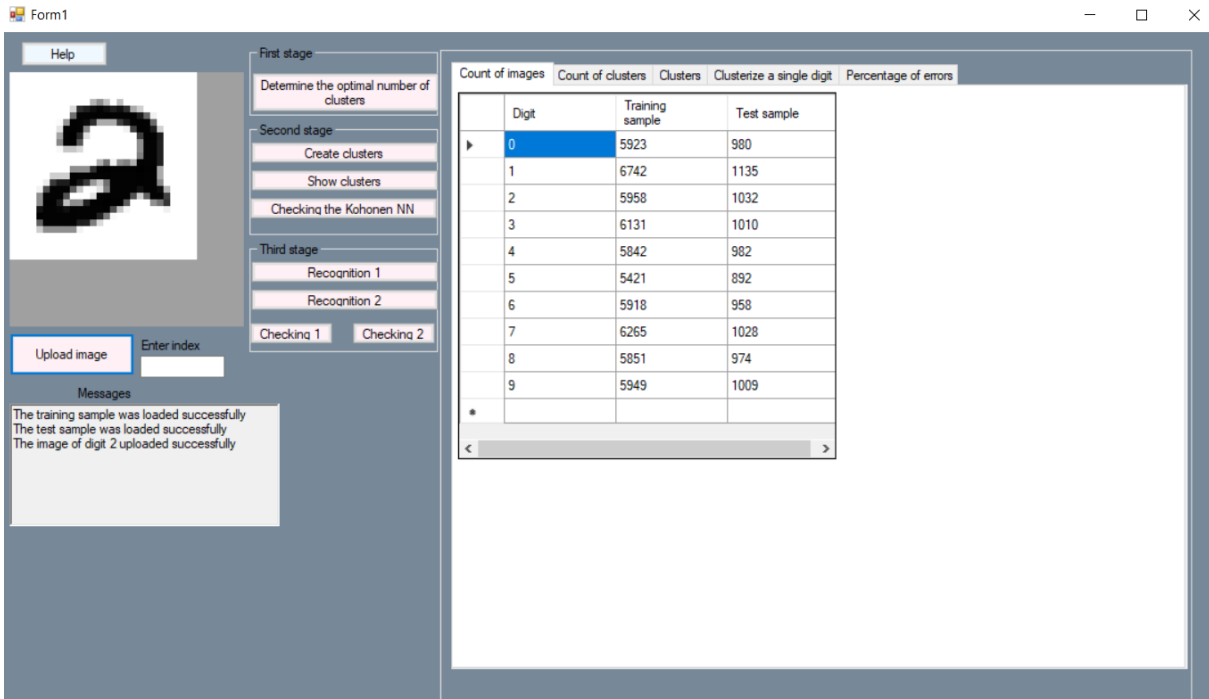


Fig. 2.3: The main window of application

- RichTextBox – for writing messages about process;
- PictureBox – for loading a random image from a test sample of the MNIST database;
- TabControl – to provide basic information about the database. Clusters and network errors. The table below shows the tabs that this element contains.

Name of tab	Description
Count of images	Contains a dataGridView that provides information about the number of images for each digit for the MNIST database.
Count of clusters	Contains a dataGridView that contains the optimal number of clusters less than or equal to 50 for each digit (for a Hopfield neural network). The tab also contains a comboBox and a "Graph" button. Using these controls, you can select a number from 0 to 9 and the chart element will show the graph of how the optimal number of clusters changes depending on a fixed percentage.
Clusters	Contains a PictureBox, that show the images of the clusters.
Clusterize a single digit	Contains a PictureBox and a button. When you

	click on the button, the uploaded image is recognized by the Kohonen neural network and the result is displayed on PictureBox.
Percentage of errors	Contains a DataGridView that presents, for each digit, the percentage of how correct were the clusters constructed by Kohonen neural network..

Table 2.2: Description of tabs in TabControl

- Buttons: the buttons are divided into stages (the stage of determining the optimal number of clusters, the stage of building clusters, and the stage of recognition by the Hopfield neural network). Also there is a button to download image from test sample.
  - “Determine the optimal number of clusters” – after clicking, a message will pop up and give you a choice: perform automatic clustering again or read the results from the file (for the MNIST database, the data is already in the file, so you can use it to speed up). You can see the results in TabControl in the "Number of clusters" tab. The tab also contains tools for plotting the dependence of the number of clusters on a fixed percentage of the maximum distance between vectors in the training sample.

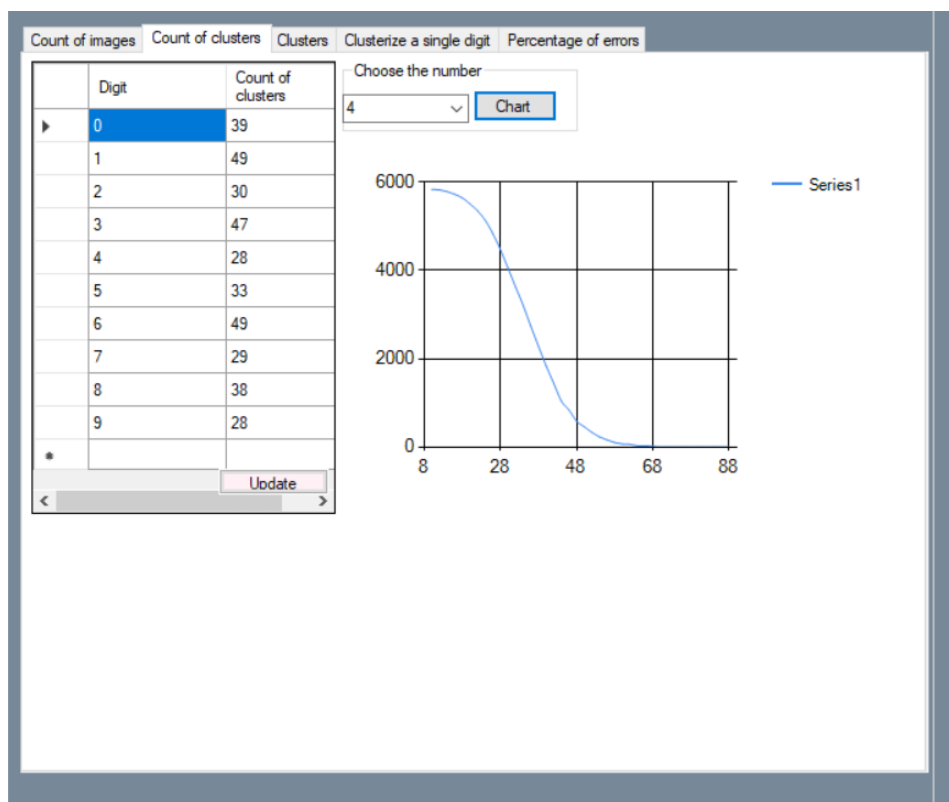


Fig. 2.4: Tab “Count of Clusters”

- “Create clusters” – after clicking, a message will pop up and give you a choice: train the data and identify the main clusters, or read the results from the file (for the MNIST database, the data is already in the file, so you can use it to speed up).
- “Show clusters” – after clicking, a window will pop up – there you need to enter the number for which we want to see the examples of clusters. You can see the results in the "Clusters" tab.

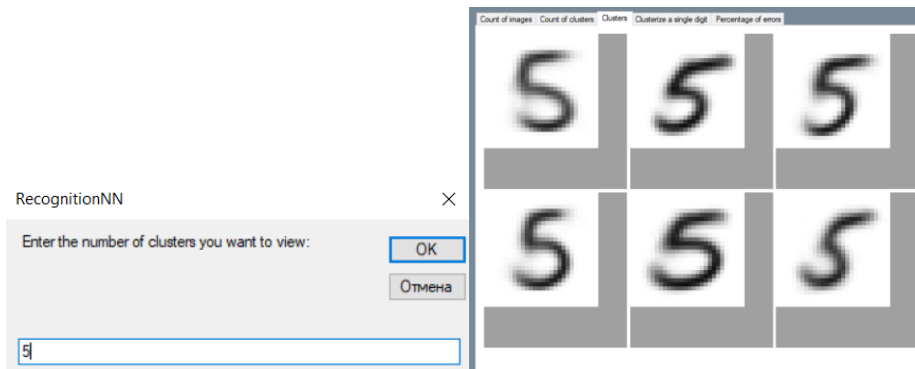


Fig. 2.5: Tab “Clusters”

- “Checking the Kohonen neural network” – after clicking, a message will pop up and give you a choice: check the data again or read the results from the file. The result will be a table that contains information about how many images from the training sample were included in the "own" cluster as a percentage.

Digit	Percentage
0	98.16139 %
1	99.82363 %
2	93.98642 %
3	94.15263 %
4	91.94699 %
5	93.04153 %
6	97.59666 %
7	91.33398 %
8	93.52518 %
9	90.67461 %
*	

Fig. 2.6: Tab “Percentage of errors”

- “Recognition 1” – after clicking, a new window that contains three PictureBox opens. The first PictureBox is the image that best matches

the image that needs to be recognized. The second and the third PictureBox are images that rank the second and the third in recognition. Recognition was performed applying a Hopfield neural network using method 1. Another three PictureBox stack the image that came out as a result and the one that should have come out on the top of each other and are highlight them in different colors, so it is easier for the user to compare them. The window also contains information about how wrong the neural network was during recognition. The dataGridView displays the percentage separately for each digit.

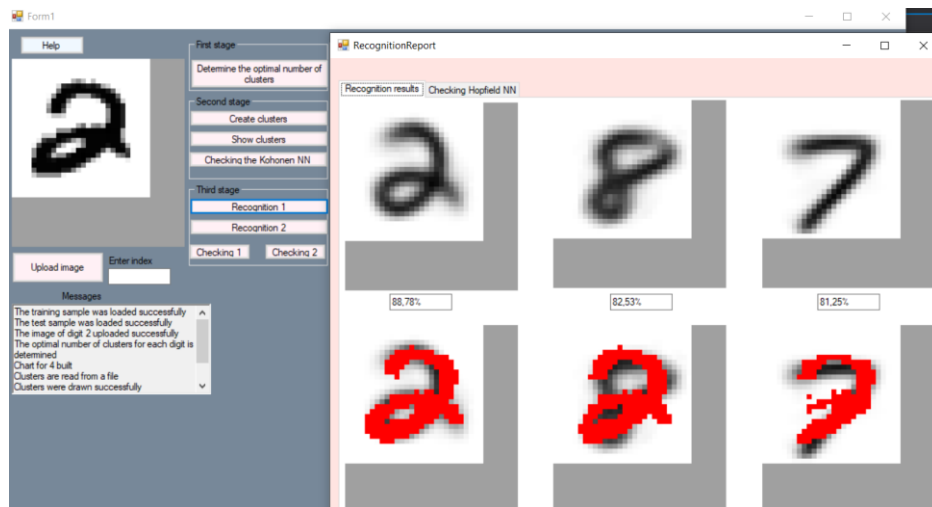


Fig. 2.7: Recognition using method 1”

- “Recognition 2” – after clicking, a new window that contains three PictureBox opens. The first PictureBox is the image that best matches the image that needs to be recognized. The second and the third PictureBox are images that rank the second and the third in recognition. Recognition was performed applying a Hopfield neural network using two methods. Another three PictureBox superimpose the image that came out as a result and the one that should have come out on the top of each other and highlight them in different colors, so it is easier for the user to compare them. The window also contains the information about how wrong the neural network was during recognition. The dataGridView displays the percentage separately for each digit.

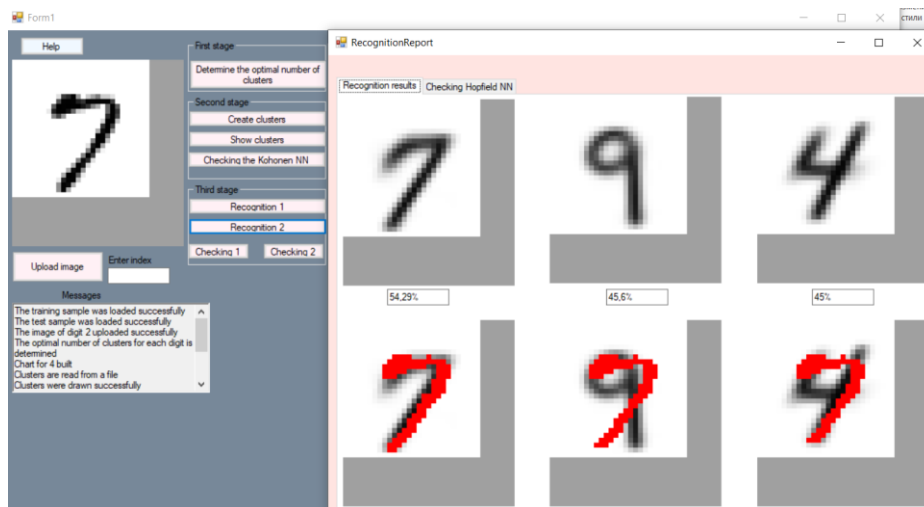


Fig. 2.8: Recognition using method 2”

- “Checking 1 “– after clicking, a message will pop up and offer you to check the data again or read from the file. It enters information about how well a Hopfield neural network works using method 1 in the table. The result will be a table that contains the information about how many images (as a percentage) from the training sample were recognized correctly.
- “Checking 2 “– after clicking, a message will pop up and offer you to check the data again or read from the file. It enters the information about how well the Hopfield neural network works using method 2 in the table. The result will be a table that contains information about how many images (as a percentage) from the training sample were recognized correctly.
- “Upload image” – after clicking on it, a random image from the test sample is loaded. it is also possible to load the image by number (numbering starts from 0).
- “Recognize” – the button is located on the "Clusterize a single digit" tab. After clicking on it, the image uploaded to the software is recognized by the Kohonen neural network.



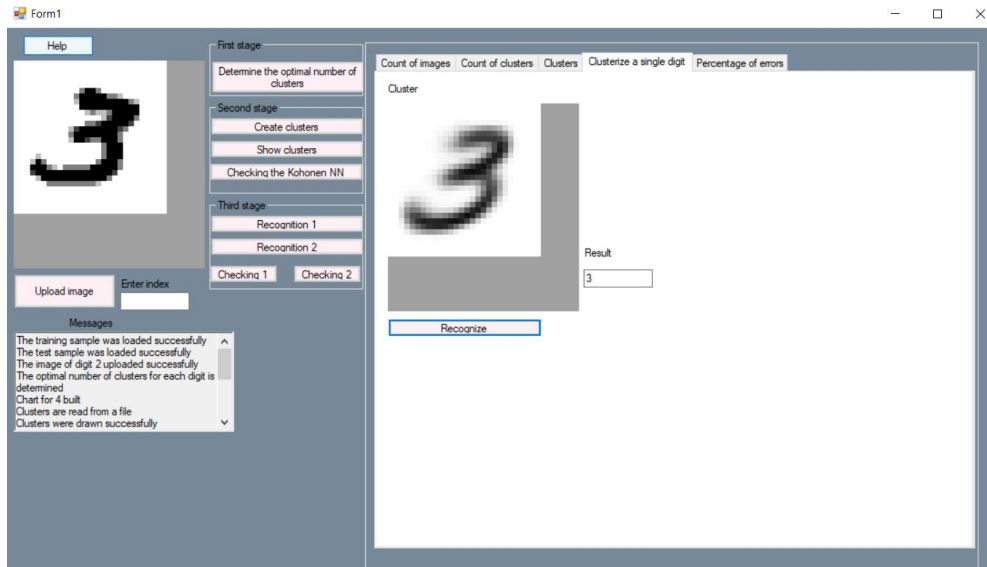


Fig. 2.9: Recognition using Kohonen Neural Network”

Before starting work, the user can read the instructions for applying this program. The user can use the «Help» button to get a detailed user guide ( \*chm file).

The application is written in C# in Windows Forms in Visual Studio 2017. Below is a sequence diagram that shows how the main application blocks interact.

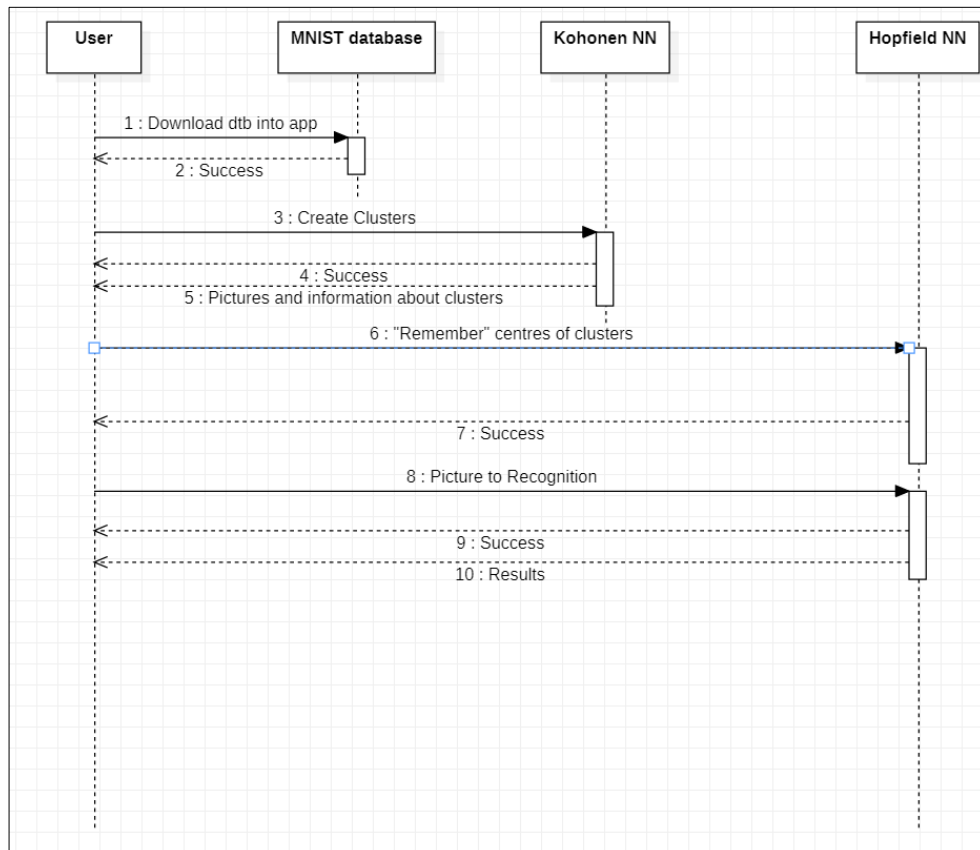


Fig. 2.10: Sequence diagram

The user first accesses the image database and uploads it to the app. In case of success or error, the user receives a message about this. Then the life cycle for the database dies, and the

application no longer interacts with it. Next, the user sends a message to the Kohonen neural network to build clusters. After the neural network has completed its work, the user receives a success message and the clusters built. Kohonen neural network completes its life cycle. Next, the user sends the received clusters to the Hopfield neural network. The neural network sends a success message to the user after it has "memorized" the cluster centers. The user then sends the image to the Hopfield neural network to be recognized. Hopfield neural network starts working and, if no errors occur, sends a success message and results. The Hopfield neural network completes its life cycle.

The program can be divided into three major stages: determining the number of clusters separately for each digit using the automatic clustering algorithm, clustering using a Kohonen network, and direct image recognition applying a Hopfield neural network. The following sub-chapters will detail these steps and the achieved results.

## 2.2 Determining the optimal number of clusters

As mentioned earlier, the MNIST database has 60,000 images in the training sample and 10,000 images in the test sample. Table 2.1 contains information about how many images from the training and test sample are contained for each digit.

You need to build clusters whose centers will serve as a memory sample for Hopfield neural network. You also need to perform cluster analysis for separately each figure.

The main problem is that all people write differently. There are many ways to write a particular number. Each number has its own specific spelling features. For example, one person will write the number 0 with a tilt to the right, another one – to the left, and the third person will write it in the center. For each digit, the number of attributes that can be used to split the sample is different. Therefore, you cannot specify the same number of clusters to split each digit into. You need to use an algorithm that automatically determines how many elements the training sample should be divided into for each digit. Thus, the task is to determine the optimal number of clusters for each digit.

In this program in order to determine the optimal number of clusters, it is necessary to press the button. The MNIST database has already been tested, so information about the number of clusters is already contained in the files. When you click the button, the program asks whether to perform clusterization again or read from the file.

The automatic clustering algorithm used in this work was discussed in detail in the previous chapter. For this algorithm, the program has created a separate class that contains a method of calculating the Euclidean distance between vectors, a method for training, and a method of writing clusters to a file. This class is inherited from the distance interface, which contains a method of Euclidean distance.

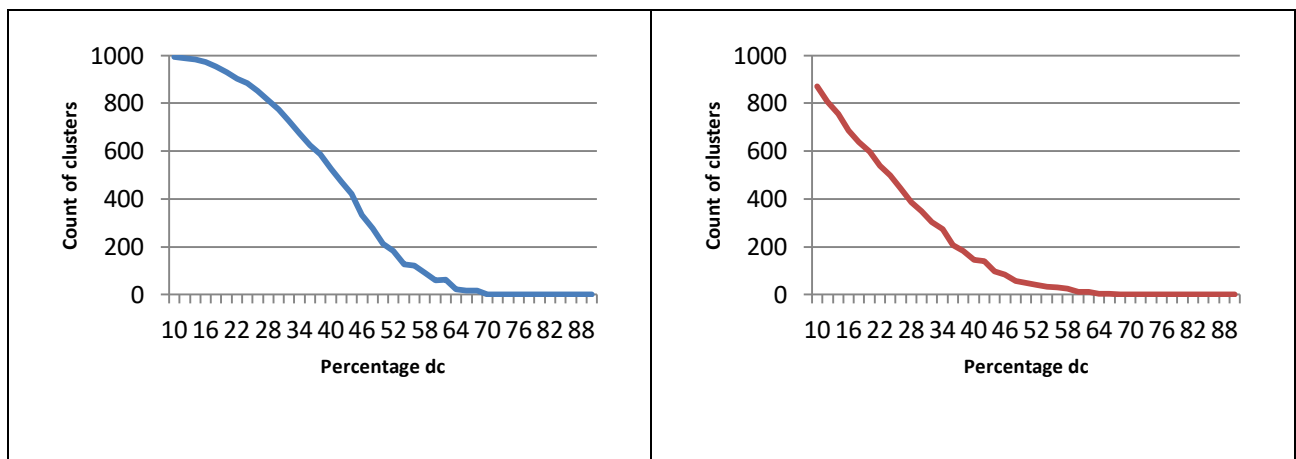
Automatic Clusterization
+sizeOfVector +countOfNumbers
+EvclidDistance(int vecorNumber, double[, ] t, double[, ]d, int vectors) +Trainig(double[, ]pattern, int vectors, int num) +ComputeAndWritingIntoFile(string fileName, double maxDist, double[, ] distance, int vectors)

Fig. 2.11: Class for Automatic Clusterization

The results are displayed in a table. The Autoclasterization class contains a constructor that fills in the fields sizeOfVector – vector dimension (image width multiplied by image height) and countOfNumbers – number of digits (10). Data is filled in from the class for describing the database.

The algorithm consists of the following steps. You must calculate the squares of the distances from each vector to all other vectors in the training sample for each digit. For each digit, a table of distances between all vectors is obtained. Next, find the maximum value in this table. The next step is to determine the maximum distance between the vectors. This will be a fixed percentage (dc) of the maximum value in the table that was obtained in the previous step. Next, you need to go through the entire table and combine the vectors in the cluster: if the distance is less than the distance obtained in the previous step, then the vectors belong to the same cluster, otherwise a new cluster is created. In this work, you just need to calculate how many such clusters you will get.

The work analyzes how the number of clusters changes depending on a fixed percentage of dc. The percentage of dc changed in the range [10;90]. We will conduct two studies. To begin with, we will take a random sample of 1000 elements for each number. A dependency graph was constructed for each number.



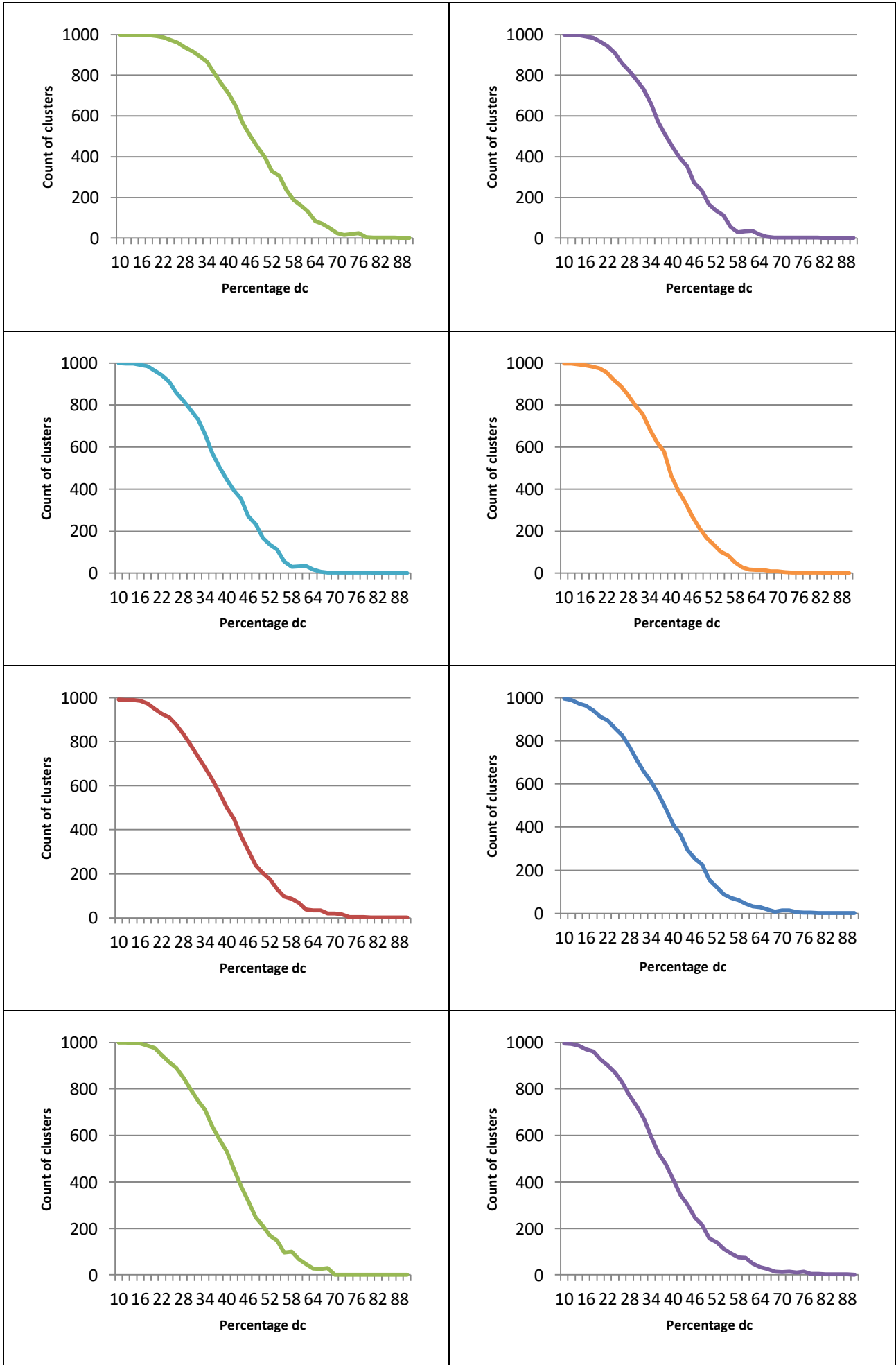


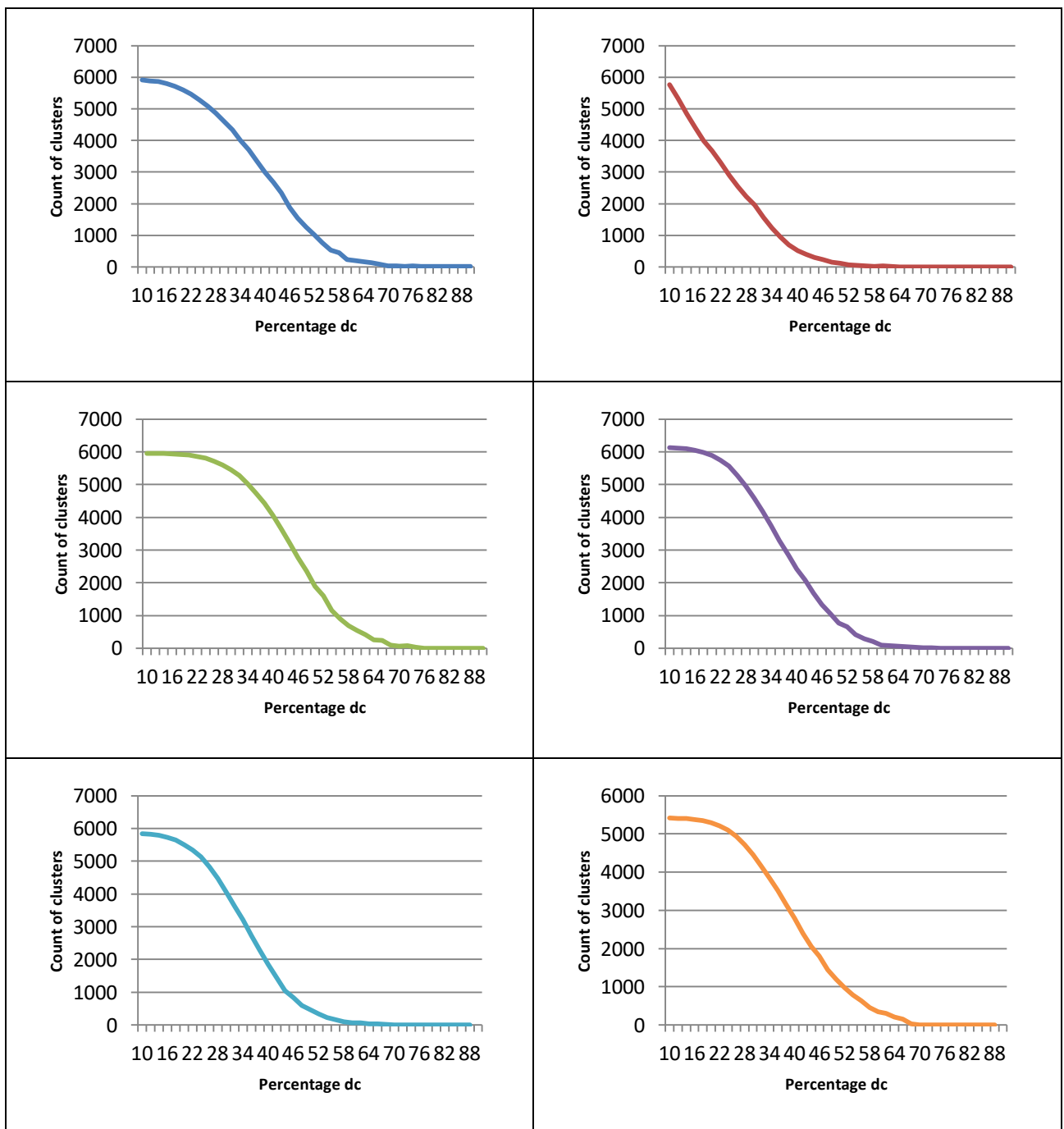
Table 2.3: The dependence of the optimal number of clusters on a fixed percentage of the maximum distance for each digit for a sample of 1000 vectors

For each digit, the graphs are approximately the same. The smaller the percentage of the maximum distance between vectors, the more clusters you need to split the sample into. We need the number of clusters to not exceed 50. The following results were obtained using this algorithm:

Digit	0	1	2	3	4	5	6	7	8	9
Clusters	22	50	48	30	29	39	44	45	46	49

Table 2.4: The optimal number of clusters for each digit with a sample of 1000 vectors

Now we will do the same, but the entire sample will be taken for each digit. As you can see in Table 2.1, the sample size is different for each number.



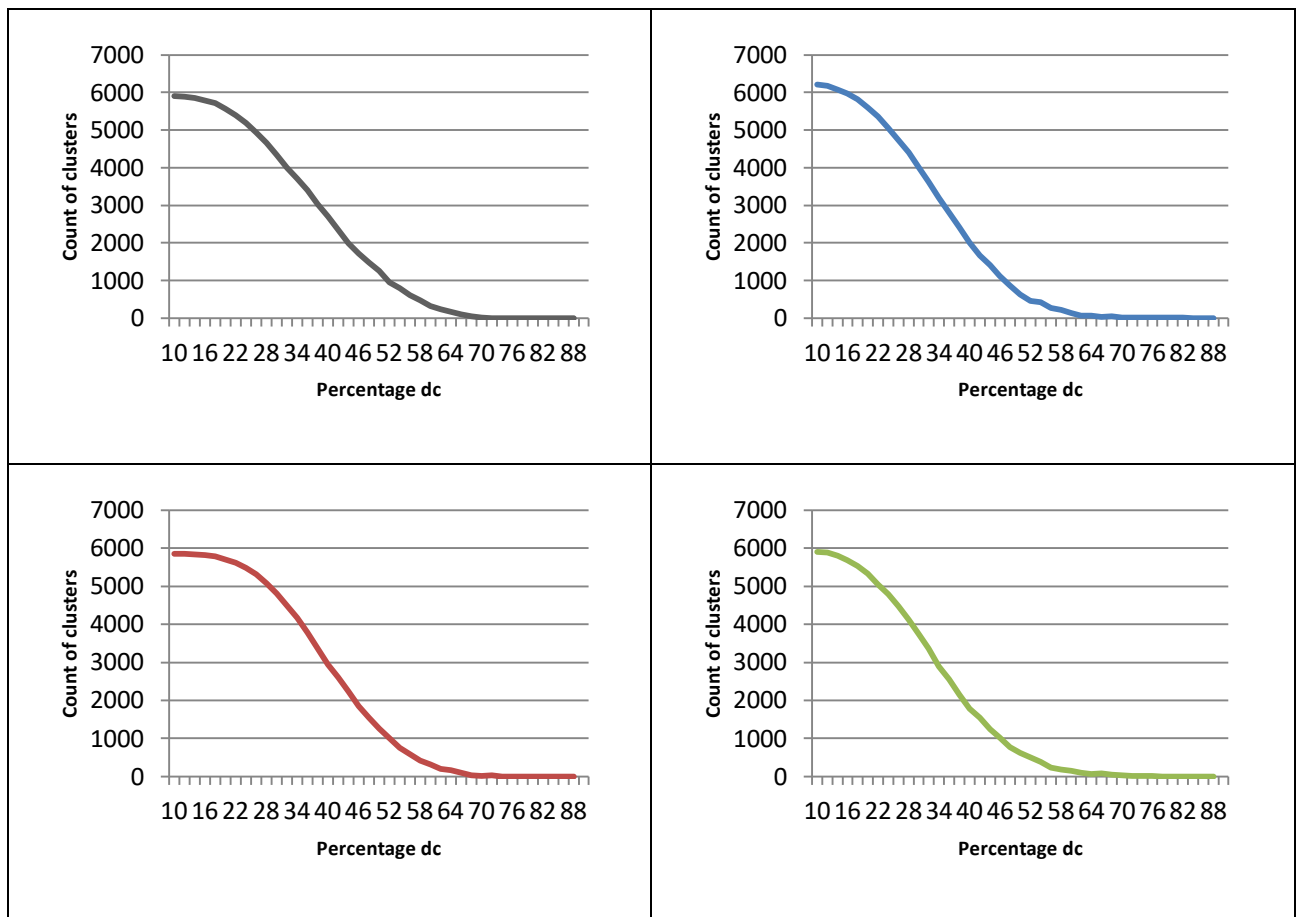


Table 2.5: The dependence of the optimal number of clusters on a fixed percentage of the maximum distance for each digit for a full sample

The optimal number of clusters in this case is obtained as follows:

Digit	0	1	2	3	4	5	6	7	8	9
Clusters	39	49	30	47	28	33	49	29	38	28

Table 2.6: The optimal number of clusters for each digit with a full sample

## 2.3 Data clustering and cluster analysis

The next step is clusterization. The training sample for each digit must be divided into the optimal number of clusters that were obtained in the previous step. Clustering was performed using a Kohonen neural network. The neural network algorithm was discussed in detail in the previous chapter.

For this algorithm, a separate class has been created that contains the main network settings, methods for training, finding the distance between the vector and clusters, and determining the minimum distance.

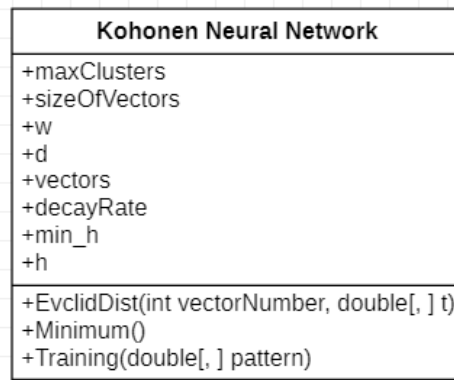


Fig. 2.12: Class for Kohonen Neural Network

The class for the Kohonen neural network uses the class for the database to get a training sample, and the class for determining the optimal number of clusters, which was described in the previous paragraph.

To cluster digits, select the following parameter values: decay Rate=0.96, min\_h=0.01, and set the initial value to h=0.6.

The results of this work are the cluster centers, which are recorded in the file. Then, using cluster centers, cluster analysis is performed. To begin with, we consider the issue of intersecting clusters with each other. By examining the intersections of clusters, you can answer the question whether clustering by the Kohonen neural network is suitable for pattern recognition. The fewer intersections between clusters, the more accurate the recognition. Then, based on the obtained clusters, we calculate which percentage of the test sample belongs to the "own" cluster, and calculate the F-measure for each digit. Based on this data, we can draw conclusions about the effectiveness of the Kohonen neural network in pattern recognition.

As a starting, we will conduct experiments on the small samples to identify common elements and characteristic features of their writing. Firstly, we conduct a study on the dependence of cluster centers on the sample size. we consider experiments on random samples of 100, 500, 1000 and 5000 images of individual numbers. In experiments a different number of clusters is chosen for each digit. It is concluded that when there are three clusters, the number of digits in them is distributed approximately equally. Based on this, for further research we assume the presence of three clusters for all numbers.

Also, the results of experiments indicate that, the centers of the clusters remain virtually unchanged starting with a sample of a thousand images. Thus, it can be assumed that the sample of 1000 elements is representative. For each digit, the images are divided into three clusters. Cluster centers are shown in Table 2.7.


Table 2.7: Cluster centers for each digit when selecting 1000 images and dividing by 3 clusters

During the analysis of each cluster, several main features can be picked out:

1. Each digit has a cluster with a strong inclination to the right.
2. There are no clusters with a clear inclination to the left.
3. All numbers have at least one cluster with no slope.
4. Images of numbers 0, 3, 4 and 5 have a wide spelling.

We can also highlight the features of writing individual numbers:

1. The third cluster of number 1 has a curl in front, unlike the other two clusters.
2. Two clusters of number 2 have a curl from at the bottom, one cluster has no curl.
3. The third cluster of number 7 has a horizontal line in the middle.
4. The second cluster of 5 contains a long horizontal line.

Note that the resulting clusters have large radius. This is because the number of clusters is small – 3. In this case, intersection of clusters may occur.

The main problem in image recognition is that identical numbers are written differently. This leads to the fact that either the number of clusters increases significantly, or the size of clusters grows. On the other hand, there is a problem that different numbers are written in a similar way. In the case of a small number of clusters, one should expect that, due to their large size, there will be intersections of clusters of different numbers. We investigate similar intersections for all images of numbers.

To build intersections between clusters, we created three tables for each image. For example, we take a sample with a volume of 1000 images 0. Next, we check at what distance from the center of each cluster for each digit is the image 0 from the sample. If the distance of the image is less than the radius of the cluster, then we assume that the image lies in this cluster. As a result, we build a table of relationships. We build similar tables for samples of all numbers. Table 2.8 shows the intersections of the clusters.



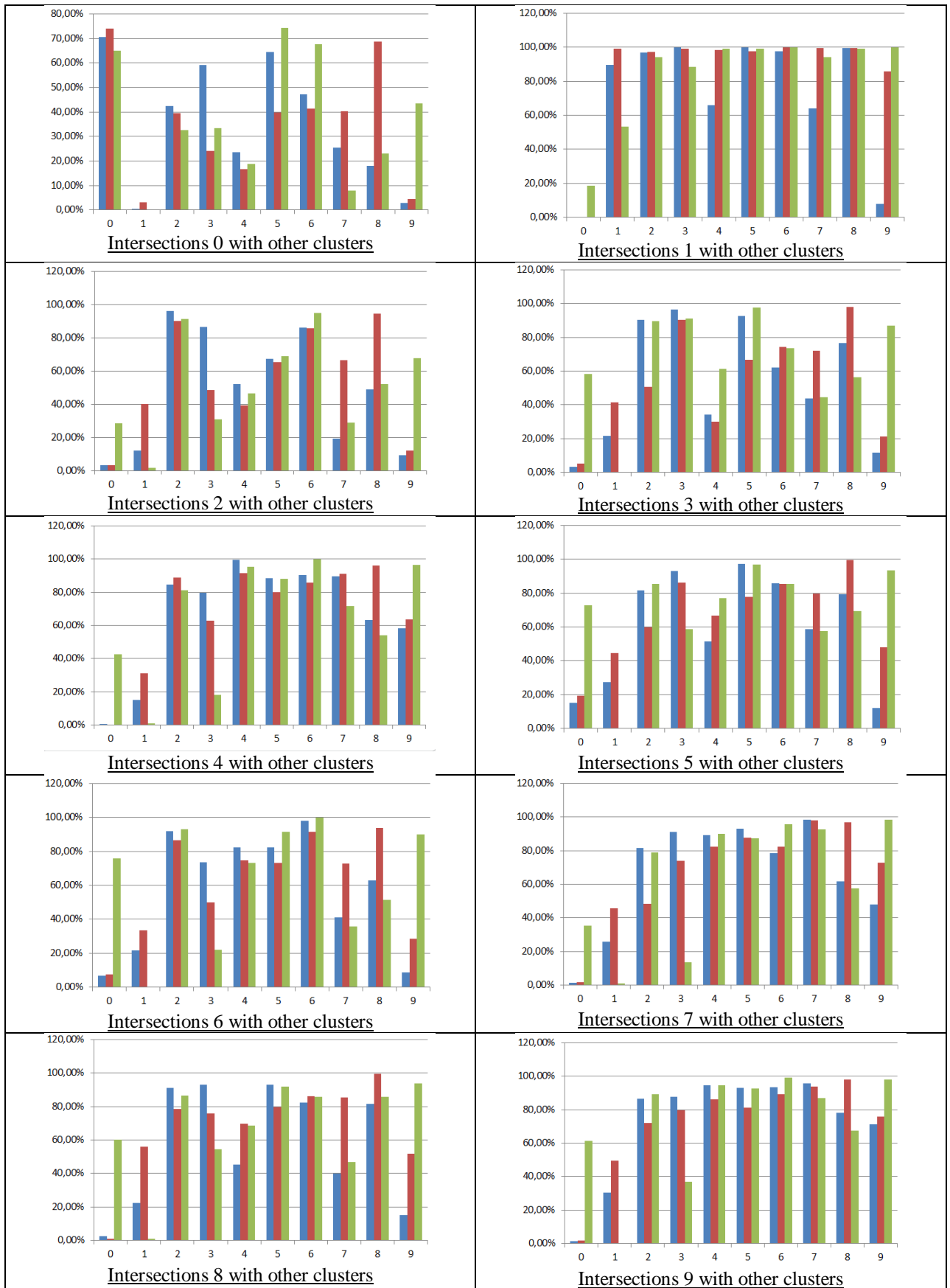


Table 2.8: Intersections of clusters when selecting 1000 images and dividing into 3 clusters

According to the Table 2.8, we made the following conclusions. Large intersections (many objects) are contained in the clusters that have the image of number 5 in them. Consequently, many

handwritten numbers are similar to the image 5. Also, it is revealed that images of number 1 have intersections with many clusters (except clusters for 0). From this we can conclude that the image of number 1 can be incorrectly recognized. Small intersections are observed in clusters for the images of number 0, which means that the image of number 0 is recognized quite accurately.

We give examples of images that are included in the intersection of clusters.

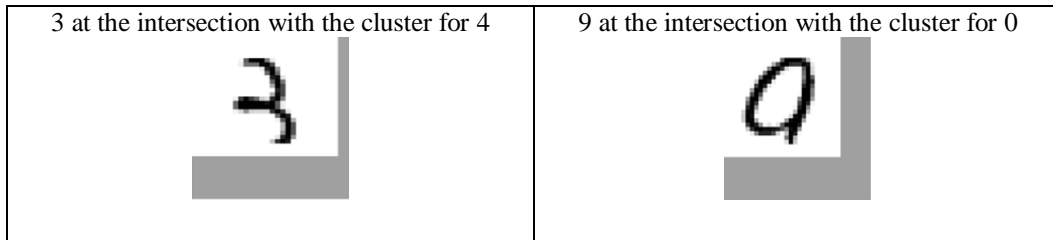


Fig. 2.13: The examples of images included in the intersection of clusters

Next, we will check the accuracy of cluster construction. To do this, for a test sample of the MNIST database, we will estimate the percentage of images contained in the desired cluster. To do this, measure the distance from each image to all the resulting clusters for each digit. If the image vector is the closest one to the cluster that belongs to the cluster group for the same number, then we assume that the vector is in the correct cluster. Fig. 2.14 shows the percentages of images from the test sample being included in clusters of their digits.

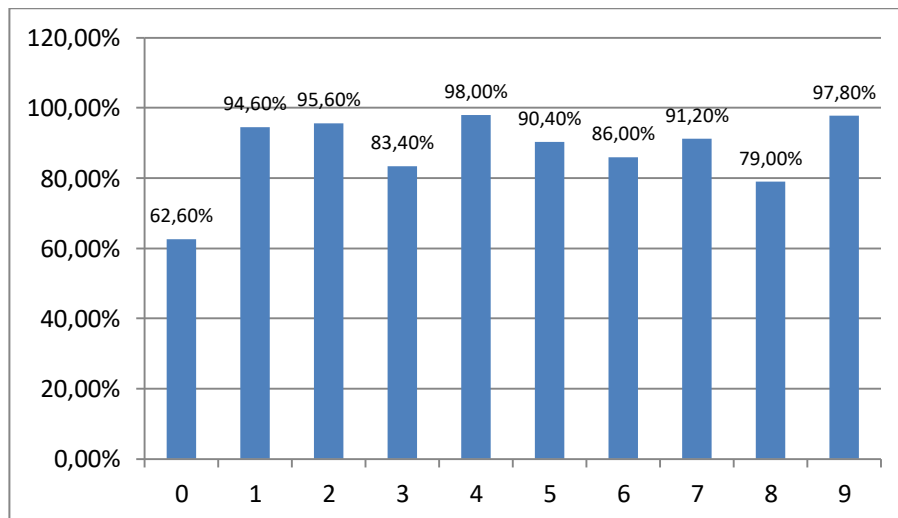


Fig. 2.14: The percentage of numbers from the test sample in its cluster

Analyzing Fig. 2.14, we can see that the images of number 0 significantly less frequently falls into the desired cluster. This is due to the fact that the test sample contains the images 0 that differ in appearance from those images that are contained in the training sample, and, therefore, are very far from the centers of the desired clusters. In Fig. 2.13 the examples of such images are presented.

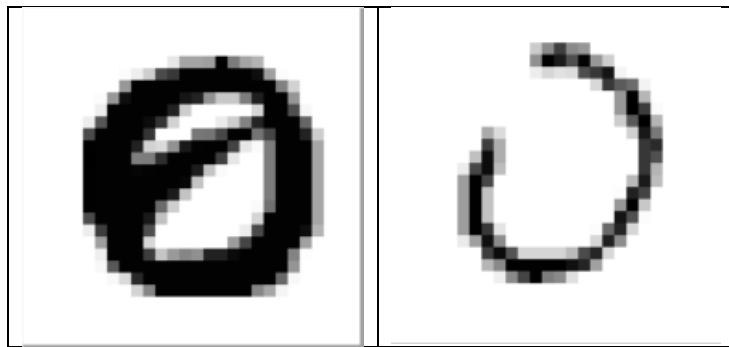


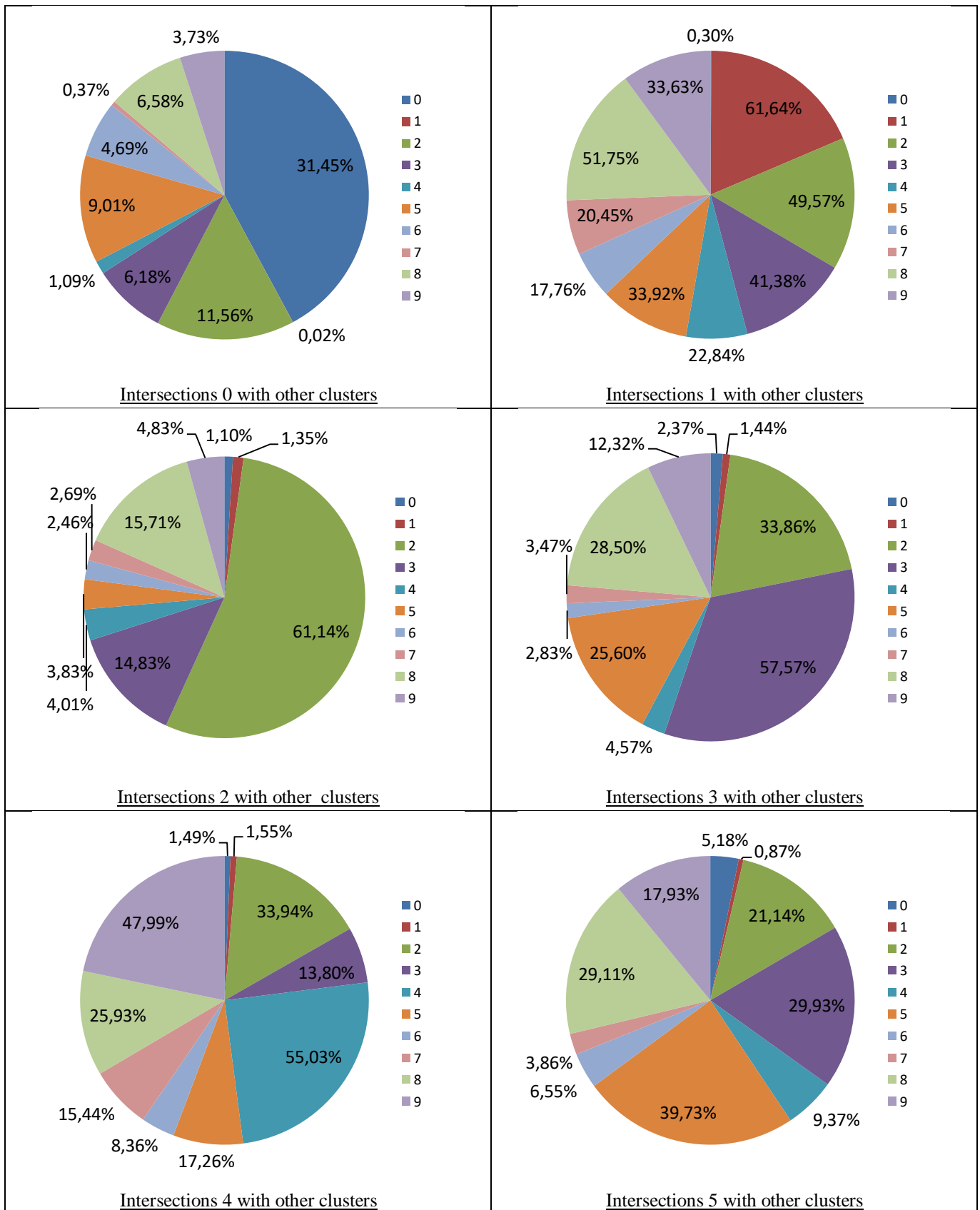
Fig. 2.15: Examples of the image of number 0 from a test sample that did not fall into its cluster

Now we will perform clusterization on the sample in full and take the optimal number of clusters. The values for the optimal number of clusters are shown in the Table. 2.6. Since the diameter of cluster is less, then the intersections of clusters are much fewer. Therefore, recognition will be more successful. Here are the examples of new clusters for each digit.


Table 2.9: Examples of clusters with a full sample and an optimal number of clusters

Let's create cluster intersections for the full sample and for the optimal number of clusters.

Table 2.10 shows the new cluster intersections.



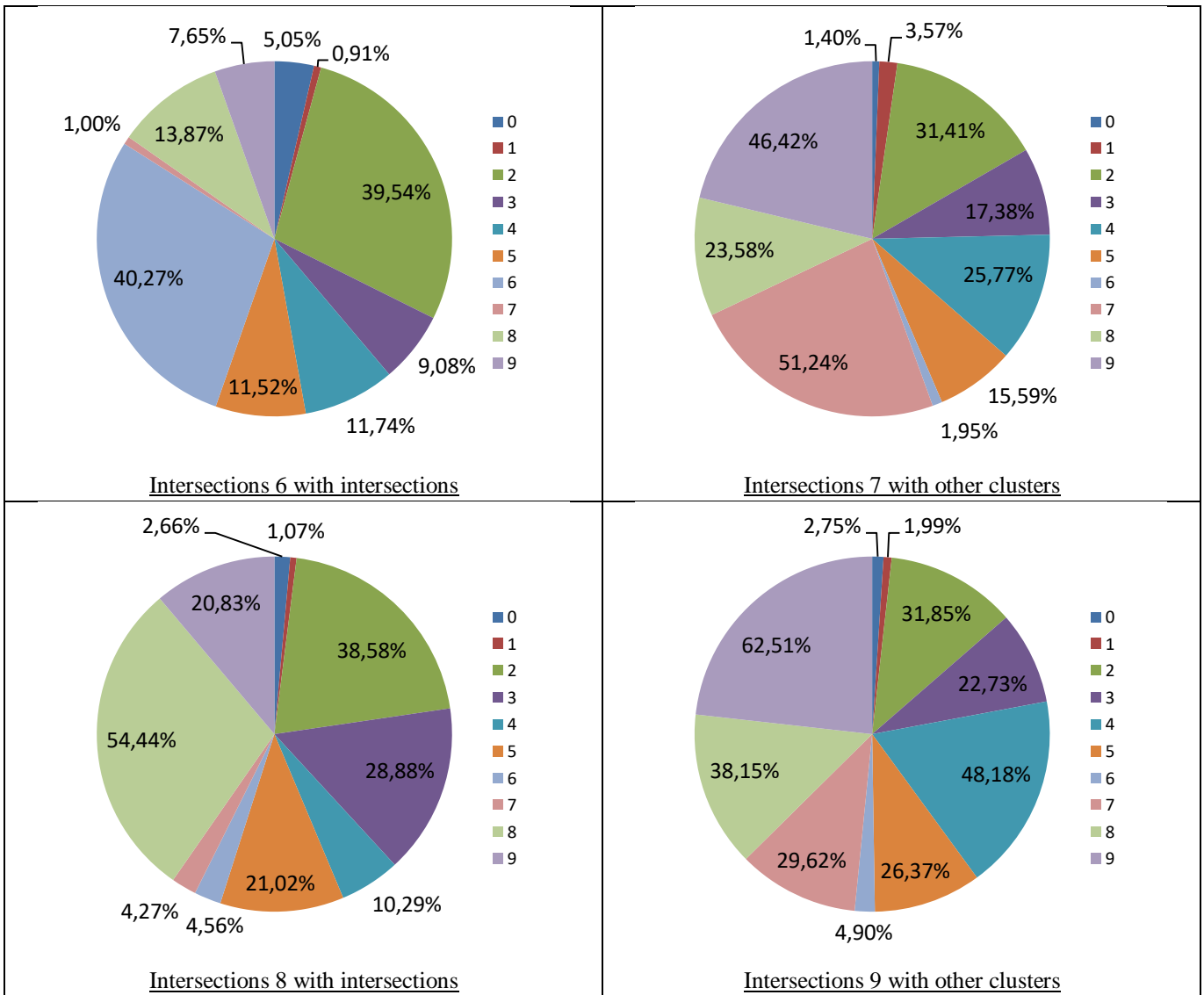


Table 2.10: Intersections of clusters

According To Table. 2.10 let's make the following conclusions. The digit 0 has intersections with clusters for the digit 2,5. This is due to the fact that these numbers have similar curves. The number 1 has more intersections than other clusters. This is due to the fact that the element of the digit 1 is contained in the writing of other digits. Clusters for digit 2 have significant intersections with clusters for digit 3,8. Digit 3 has intersections with digits 2,5,8. Digit 4 intersects with clusters for digit 9. Digit 5 intersects with clusters for digit 3,8. Digit 6 intersects with digit 2. Digit 7 intersects with digit 9. Clusters of the number 8 has intersections with clusters for the numbers 2,9. the Number 9 has significant intersections with the number 4. the Intersections are much less than in the situation when we divide into 3 clusters. This way, it greatly improves the results of the network. Here are examples of images that are included in the intersections of clusters.






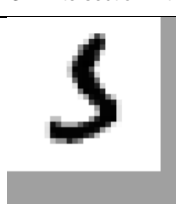

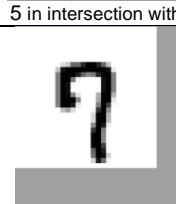
 0 in intersection with 6	 1 in intersection with 6
 2 in intersection with 3	 3 in intersection with 2
 4 in intersection with 9	 5 in intersection with 3
 6 in intersection with 2	 7 in intersection with 9

Table 2.11: Examples of images that are included in cluster intersections

In order to estimate how accurately a Kohonen neural network clusters the data, we will also estimate the percentage of images from the test sample falling into the desired cluster. Figure 2.16 shows the percentages of images from the test sample falling into clusters of their own digits for full samples and for the optimal number of clusters.

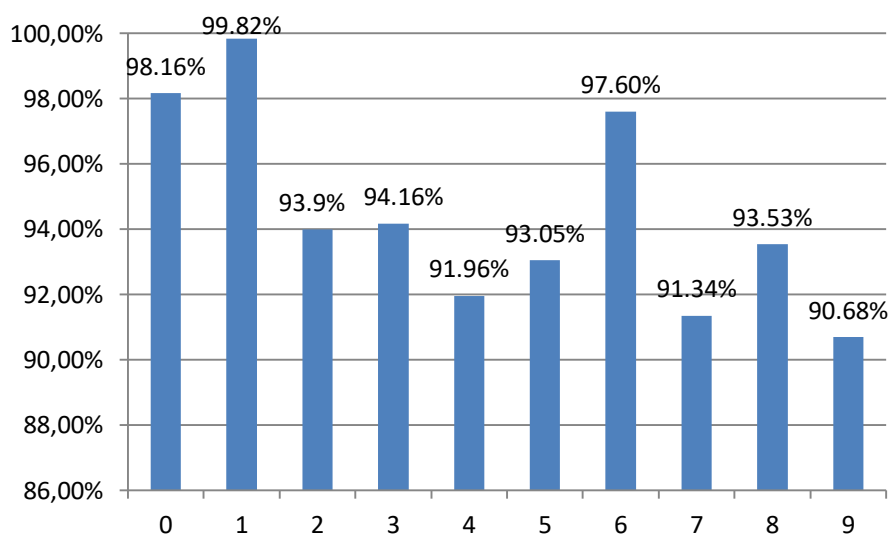


Fig. 2.16: The percentage of numbers from the test sample in its cluster

Table 2.12 shows a summary table presenting where a Kohonen neural network made errors.

	0	1	2	3	4	5	6	7	8	9
0	962	1	4	1	0	2	7	1	1	1
1	0	1133	1	0	0	0	1	0	0	0
2	9	10	970	10	1	0	3	10	17	2
3	0	1	5	951	1	19	0	7	19	7
4	0	11	4	1	903	0	11	2	6	44
5	4	1	1	25	1	830	14	2	10	4
6	10	3	4	0	2	2	935	0	2	0
7	1	23	10	2	10	1	0	939	0	42
8	4	1	4	21	3	19	3	5	911	3
9	5	8	5	8	37	2	1	22	6	915

Table 2.12: Summary results of recognition by the Kohonen neural network

Also, to evaluate the accuracy of clustering of the Kohonen neural network, we calculate Recall, Precision, and F-measure.

Precision is the percentage of objects that are called positive by the classifier and are actually positive. In other words, this is the number of vectors that a Kohonen neural network has assigned to a particular cluster and these vectors actually belong to that cluster. Recall shows the percentage of positive class objects of all positive class objects found by the classifier. In other words, this is the number of objects in a particular cluster. Recall demonstrates the classifier's ability to detect a class, and precision allows you to distinguish this class from other classes. The F-measure is calculated using the following formula:

$$F_{measure} = 2 \frac{Precision * Recall}{Precision + Recall} \quad (22)$$

Thus, the table with the F-measure has the form:

Digit	0	1	2	3	4	5	6	7	8	9
Recall	0.967	0.951	0.962	0.933	0.943	0.949	0.959	0.950	0.937	0.899
Precision	0.982	0.998	0.940	0.942	0.920	0.930	0.976	0.913	0.935	0.907
F-means	0.974	0.974	0.951	0.937	0.931	0.939	0.967	0.932	0.936	0.903

Table 2.13: F-measure

Analyzing Fig. 2.16 and Table. 2.13 it can be concluded that a Kohonen neural network can be used for recognizing handwritten digit images, since a large percentage of images from the test sample fall into the desired cluster. Successful recognition is achieved if we take a full sample and a large number of clusters. Due to the fact that there are many clusters and the cluster radius are small, there are not many intersections. Note that the number 9 is recognized the worst – just over 90 percent. And the number 1 is recognized by the Kohonen neural network in almost 100 percent of cases.

Thus, the percentage of vectors from the test sample falling into the correct cluster for each digit is more than 90%. This means that clusterization can be used for pattern recognition of digits.

Therefore, the cluster centers turned out to be correct and we can use them as objects for "memorizing" by a Hopfield neural network.

## 2.4 Pattern recognition by a Hopfield neural network

The last stage of recognizing images of handwritten numbers in the MNIST database is recognition by a Hopfield neural network. The architecture and algorithm were discussed in detail in the previous chapter.

A separate class has been created for the Hopfield neural network in the software. The class contains two methods of neural network operation, a method for storing images, in this work, these are cluster centers obtained in the previous step, and a method that describes the activation function of the Hopfield neural network. A separate Hopfield neural network is created for each digit from the MNIST database.

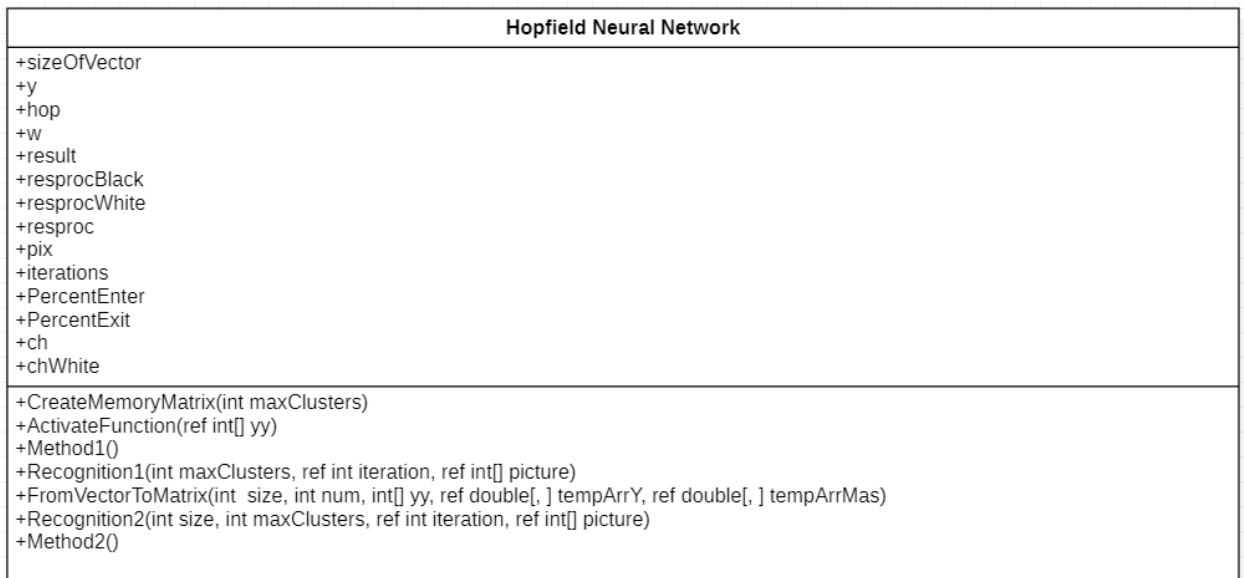


Fig. 2.17: Class for Hopfield Neural Network

The Hopfield neural network class uses clusters obtained using the KohonenNeuralNetwork class and the image vector to be recognized using the database class. The parameters used in this network were selected experimentally: pix=100, iterations=200, PercentEnter=4, Percent=2, ch=15, chWhite=3.

The result of the network is a vector image that either matches the vector from memory after a certain number of iterations, or the most similar vector is selected. Each of the neural network methods describes a way to determine two similar image vectors. The network result is given with a percentage of how much the original image matches the image from the neural network memory. After all the networks (for ten digits there will be 10 networks) have completed their work, as the final result, 3 images are selected that most closely match the original image, that is, they have high percentages. To do this, the results of all networks (percentages) are sorted in descending order and



the first 3 vector images are taken.

The resulting images are output in a separate form. Each image also displays a separate image with an overlay of the original image. This is done in order to clearly demonstrate where the original image and the result have similarities.

As mentioned in the previous chapter, if the outputs of the Hopfield neural network have stabilized, but the output values do not match any vector from the network memory, then the response is the sample that best matches the input vector. Let us go to the description of the two methods used in this work and compare the results.

The first method is based on a simple comparison of the image under study and the image stored in the network memory. At the last step in a Hopfield algorithm, you need to check how much the sample to be recognized resembles the k-th sample in memory. Verification is performed using the following formula:

$$C = C + 1, \text{ if } (y_j = x_j^k = 1), \quad (23)$$

where C is the number of pixels. As can be seen from formula (23), the check is performed only on matched 1s, since it is the value of 1 that makes a greater contribution to the samples. For example, if you have an image recognition task, -1 is the white pixels, and 1 is the black pixels. Thus, the black pixels are very important.

After the number of matching values equal to 1 is calculated, this value is recalculated as a percentage, that is, divided by the dimension of the vector. In the case of the MNIST base, the dimension of the vector is 784 (28\*28). If the percentage is less than a certain number of Percent, the algorithm continues its work (moving to step 2 of the Hopfield algorithm). The Percentage number is selected experimentally. The choice of number depends on how many 1s (grayscale) are contained in the samples. If the number 1 is small (as in the case of number recognition), then the number Percent will also be small.

The method was evaluated on all images of the MNIST test sample. Each image from the test sample was sent to the Hopfield neural network input for each digit, and it was evaluated whether the network correctly recognized the number. Table 2.14 shows the results of this work using the first method.

0	1	2	3	4	5	6	7	8	9
74,69%	91,19%	74,42%	65,45%	79,12%	71,30%	74,01%	84,63%	46,61%	69,77%

Table 2.14: Results of recognition by the Hopfield neural network using method 1

Based on the achieved results, the following conclusions can be drawn:

- Method 1 is good at recognizing the numbers 0,1,2,4,5,6,7: more than 70%.
- Number 8 is very poorly recognized: just over 45%, i.e. more than half of these numbers are recognized as other numbers.

Table 2.15 shows a summary table showing where the Hopfield neural network made errors using method 1.

	0	1	2	3	4	5	6	7	8	9
0	732	21	41	15	28	50	24	26	32	11
1	8	1035	2	3	22	0	0	48	14	3
2	7	153	768	13	15	7	3	61	2	3
3	0	102	14	661	9	60	1	28	16	119
4	1	100	1	1	777	2	4	12	0	84
5	3	66	1	41	41	636	8	36	22	38
6	7	117	3	2	67	34	709	15	3	1
7	1	52	21	4	23	4	0	870	1	52
8	1	166	23	83	48	56	5	46	454	92
9	2	56	3	9	148	4	2	76	5	704

Table 2.15: Summary results of Hopfield neural network recognition using method 1

Based on the data obtained from Table 2.15, it can be concluded that most often the numbers most often give the number 1 as an erroneous result. This is because the image for the digit 1 is written as a straight line that is contained in the image for each digit. Almost all images are contained in the image of the number 1. It can also be concluded that the number 8, which is recognized the worst, is most often recognized as the number 1.

Thus, the error of recognition by the neural network is 26%. Here is a table with measures.

	0	1	2	3	4	5	6	7	8	9
Recall	0,961	0,554	0,876	0,794	0,660	0,746	0,938	0,714	0,827	0,636
Precision	0,747	0,912	0,744	0,654	0,791	0,713	0,740	0,846	0,466	0,698
F-measure	0,840	0,689	0,805	0,718	0,719	0,729	0,827	0,775	0,596	0,665

Table 2.16: F-measure for the Hopfield neural network using 1 method

Here are the examples of images in which the neural network made errors and recognized the wrong image.

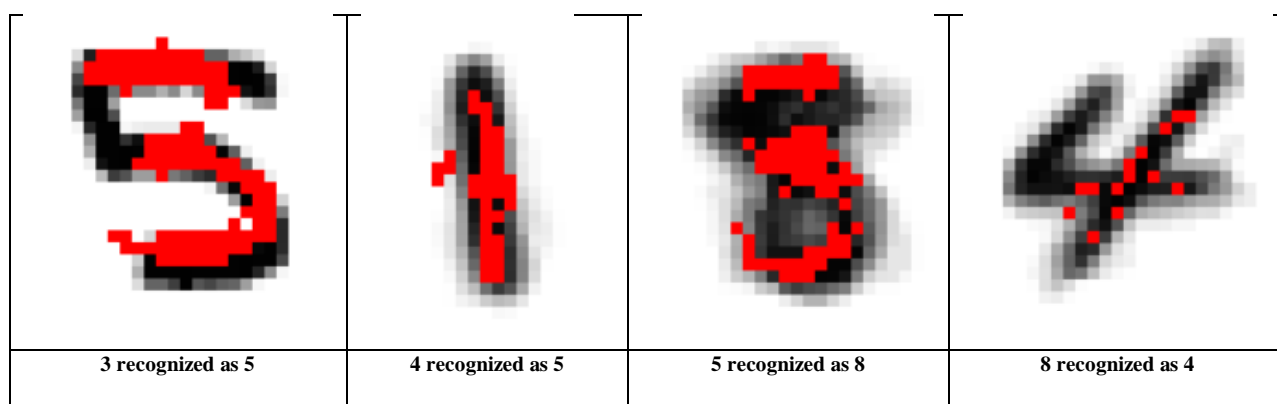


Fig. 2.18: Examples of incorrect recognized images when using method 1

Here are the examples of how the network works correctly.

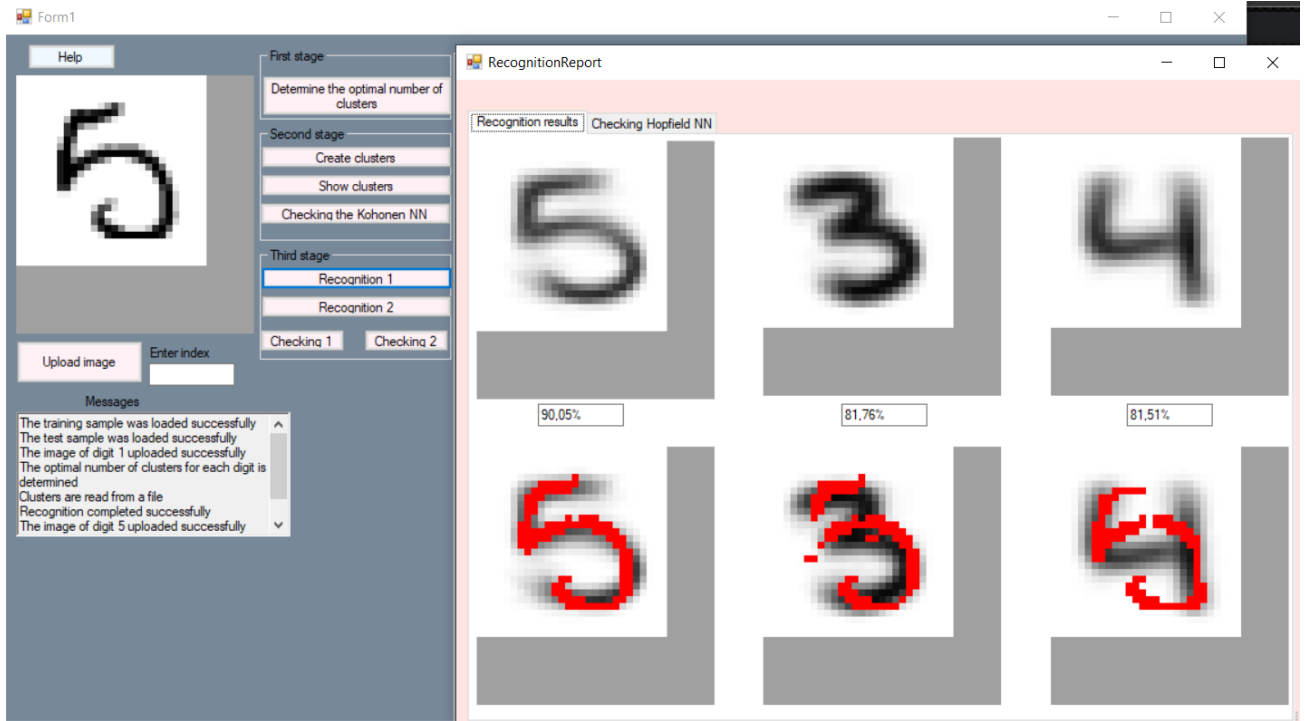


Fig. 2.19: Examples of correct recognized images when using method 1

In the second method, to find a vector from memory that is best similar to the original one, you need to represent the vector as a matrix. In this task, each vector is a 28 by 28 matrix. Next, you need to compare the matrices by rows and by columns. Next, the algorithm will be described for rows, and similar actions are performed for columns.

Each row counts the number of matched values equal to 1 and the number of matched values equal to -1. It also counts the number of values equal to 1 in the matrix that corresponds to the desired vector, and the number of values equal to 1 in the matrix that corresponds to the k-th vector from memory. Similarly, the number of values equal to -1 is calculated. Then we calculate the result of how much the matrix row corresponding to the source vector is similar to the matrix corresponding to the vector from memory using the following formulas:

$$CB = CB + 1, \text{ if } \left( \left| \left( \frac{OB}{AllYB} \right) * 100 - \left( \frac{OB}{AllMB} \right) * 100 \right| < ovB \right) \quad (24)$$

$$CW = CW + 1, \text{ if } \left( \left| \left( \frac{OW}{AllYW} \right) * 100 - \left( \frac{OW}{AllMW} \right) * 100 \right| < ovW \right), \quad (25)$$

where:

- CB(CW) – the number of rows with significant matches of values 1(-1);
- OB(OW) – the number of matched 1 (-1), this value is calculated separately for each row;
- Ally (AllY W) – the total number of 1 (-1) in a row for the matrix that corresponds to the original vector;
- All MB(All MW) – the total number of 1 (-1) in a row for a matrix that corresponds to a vector from memory;

- $ovB - ovC$  - a number that indicates how similar two rows are.

The difference in formulas (24)-(25) tends to 0 – the more similar the rows are, that is, the more matches, the smaller the difference. In this paper,  $ovB=15$ ,  $ovW=3$ . As in method 1, these parameters were selected experimentally. How to choose such tasks depends on which images are in the database: what shade prevails. The number  $ovW$  is significantly less than the number  $ovB$ , since the number of white pixels in images of numbers strongly outnumbers the number of black pixels. Therefore, there will be a lot of white pixel matches. We perform similar calculations for columns.

Then, after calculating the  $CB$  and  $CW$  for rows and columns, you need to calculate the similarity of the source vector with the vector from memory using the formula:

$$Result = \frac{CBS}{NBS} * 100 + \frac{CWS}{NWS} * 100 + \frac{CBC}{NBC} * 100 + \frac{CWC}{NWC} * 100, \quad (26)$$

where:

- $CBS$  – the number of rows where 1 matches is sufficient;
- $CWS$ -the number of rows where there are enough matches -1;
- $CBC$ -the number of columns where 1 matches is sufficient;
- $CWC$ -the number of columns where there are enough matches -1;
- $NBS$  is the number of rows containing at least one 1;
- $NWS$ -number of rows containing at least one -1;
- $NBC$ -number of columns containing at least one 1;
- $NWC$ -number of columns containing at least one -1;

Then the algorithm is similar to method 1. the Results are shown in Table 2.17.

0	1	2	3	4	5	6	7	8	9
89,18%	94,63%	53,59%	56,34%	63,75%	52,80%	84,97%	65,27%	55,85%	66,11%

Table 2.17: Results of recognition by the Hopfield neural network using method 2

Based on the results obtained, the following conclusions can be drawn:

- Digits 1 are recognized more accurately than all the others.
- 2, 5 numbers are the worst recognized: just over 50% of the numbers in the test sample are recognized correctly by method 2.

Table 2.18 presents a summary table showing where a Hopfield neural network made errors using method 2.

	0	1	2	3	4	5	6	7	8	9
0	874	25	5	3	1	11	15	5	37	4
1	10	1074	1	3	5	2	7	10	21	2
2	124	100	553	58	5	64	79	23	23	3
3	50	46	64	569	14	109	26	40	75	17
4	31	112	10	9	626	6	11	6	24	147

5	95	47	23	94	15	471	45	23	66	13
6	50	69	7	1	1	5	814	0	11	0
7	10	117	9	32	52	4	1	671	21	111
8	149	118	6	39	8	49	9	34	544	18
9	29	53	0	16	140	8	3	74	19	667

Table 2.18: Summary results of Hopfield neural network recognition using method 2

Based on the data obtained from Table 2.18, it can be concluded that the following errors are often made when a Hopfield neural network recognizes using method 2:

- Digit 0 is recognized as digit 8
- Digit 8 is recognized as digit 0
- Digit 4 is recognized as digit 9
- Digit 9 is recognized as digit 4
- Digit 1 is recognized as digit 7
- Digit 7 is recognized as digit 1,9

These results are explained by the fact that these numbers contain part of each other, that is, images of numbers have common features. Thus, the neural network's recognition error is 32%. Calculate the F-measure.

	0	1	2	3	4	5	6	7	8	9
<b>Recall</b>	0,615	0,610	0,816	0,691	0,722	0,646	0,806	0,757	0,647	0,679
<b>Precision</b>	0,892	0,946	0,536	0,563	0,637	0,528	0,850	0,653	0,559	0,661
<b>F-measure</b>	0,728	0,742	0,647	0,621	0,677	0,581	0,827	0,701	0,599	0,670

Table 2.19: F-measure for a Hopfield neural network using 2 method

Here are the examples of images in which the neural network made errors and recognized the wrong image.

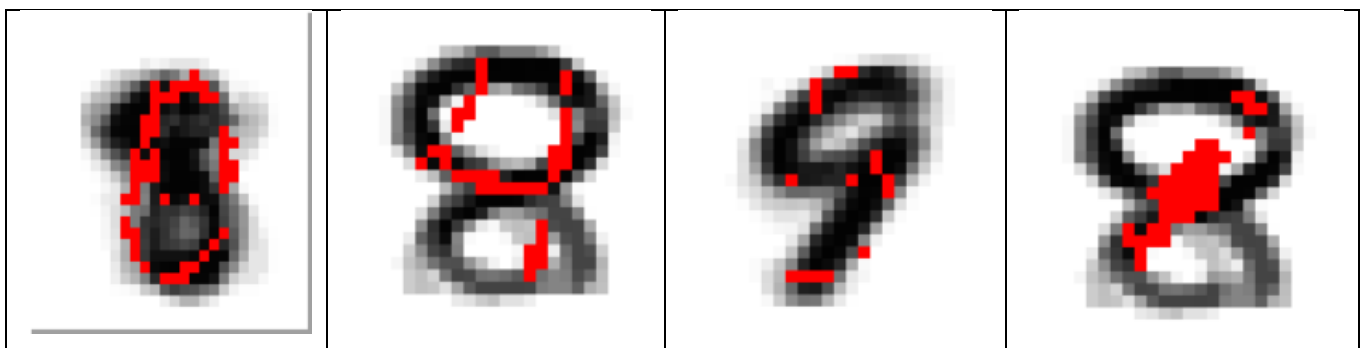


Fig. 2.20: Examples of incorrect recognized images when using method 2

Here are the examples of how the network works correctly.

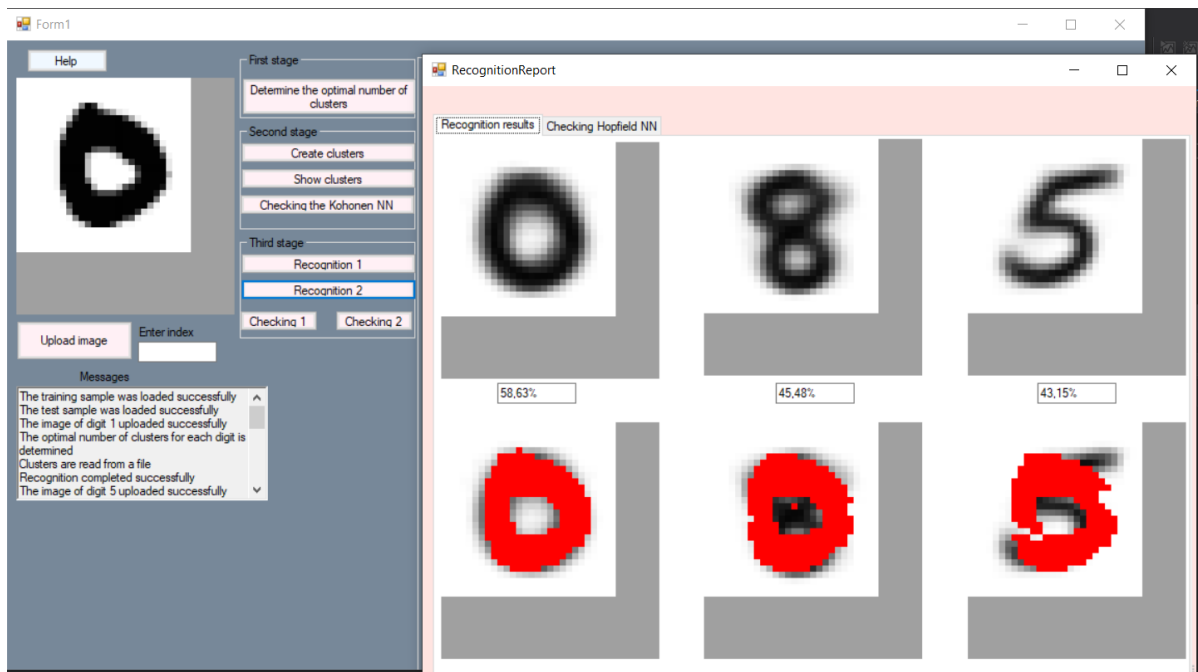


Fig. 2.21: Examples of correct recognized images when using method 2

Thus, two methods were analyzed to compare 2 images. As an output, we can say that method 1 is more suitable for pattern recognition than method 2. Errors in the recognition of handwritten numerals is that when using the f-metric is not taken into account that the images are shifted, i.e. different images have the different inclination and may be located above, below, right or left.

## 2.5 Parallel implementation of learning of the Kohonen neural network

The process of training a neural network is quite time-consuming. Therefore, parallelization of learning networks using CUDA technology is becoming more popular. This is primarily related to the uniformity of operations performed during training. In [17], CUDA was used to detect a text using neural networks and a 15-fold acceleration was received. In [18], CUDA technology was also applied in the implementation of self-organizing maps based on MapReduce that performs related calculations. In this work, the cores are optimized to provide unified memory access and efficient use of shared memory. A tenfold acceleration was obtained. In [19], we considered the Batch-SOM algorithm that is used for clustering. CUDA technology was also used here for parallelization. Parallelization took place at the level of the training sample and was accelerated by 11 times. In [20], the HPSOM algorithm was considered, that was parallelized using CUDA and MPI technologies. it was shown that it is possible to achieve not only good results in speed, but also optimize memory costs. In [21], it was found that there is a decrease in performance for the OpenCL implementation compared to the CUDA technology. Also in this work, using various combinations of OpenCL, CUDA

for two different video cards, the possibility of scaling the implementation of the SOM algorithm on multiple graphics devices was demonstrated. In [22], we also obtained acceleration of the SOM algorithm on GPUs.

As mentioned earlier, the process of training a neural network takes quite a long time. For each digit you need to train a separate Kohonen neural network. The training samples contain about 6000 images, the dimension of the input vector is 784, and the number of clusters for each digit varies from 28 to 49 (see Table 8). as a rule, network training takes several dozen iterations. In our case (the parameters selected in the previous paragraph), the training algorithm contains 84 iterations. Let us parallelize the neural network training algorithm.

To compute the execution time of the program, the clustering stage of the Kohonen neural network must be rewritten in C++. This is because CUDA technology will be used for parallelization. C# is not supported by this technology. Thus, for the Kohonen neural network, a separate C++ program is written without using an object-oriented approach.

In general, there are the following levels of parallelization [16]: the learning phase level, the training sample level, the layer level, the neuron level, and the weights level.

The choice of parallelization level depends on the number of neurons, computing nodes, and features of the computer architecture of the artificial neural network. There are two long stages in the training algorithm of the Kohonen neural network.

The first stage involves calculating the distance between the weights and the vector from the training sample. The function for calculating the distance and selecting the minimum distance on the CPU is shown in Fig. 2.22.

```
void Distance(int cluster, int vectorNumber, float* x[],
float* w[], int* cl){
    float min = FLT_MAX, temp;
    int num = 0;
    for (int i = 0; i < cluster; i++){
        temp = 0.0;
        for (int j = 0; j < N; j++){
            float r = w[i][j]-x[vectorNumber][j];
            temp += r * r;
        }
        if (temp < min) {
            min = temp;
            num = i;
        }
    }
    *cl = num;
}
```

Fig. 2.22: Function that finds the nearest weight (cluster) to the specified vector

The function contains the distance from the vector of the training sample  $x[\text{vectorNumber}]$  to each of the clusters  $w[i]$ . The number of clusters is indicated by cluster. this value varies from 28 to 49, depending on the number being trained for. The complexity of this function is  $O(\text{cluster} * N)$ ,

where N is 784.

The second stage involves passing through all vectors and changing the weights of neurons. The training function has the form:

```
void TrainingSeq(int cluster, int vectors, float* w[],
float* x[]){
    float h = 0.6;
    float rate = 0.96;
    float minh = 0.002;
    int dMin;
    do{
        for (int k = 0; k < vectors; k++){
            Distance(cluster, k, x, w, &dMin);
            for (int i = 0; i < N; i++){
                w[dMin][i] = w[dMin][i]+h*(x[k][i]-
w[dMin][i]);
            }
        }
        h *= rate;
    } while (h > minh);
}
```

Fig. 2.23: Sequential learning algorithm for the Kohonen neural network

In the training function, the pass is performed on all vectors from the training sample. The number of vectors in the selection is indicated by the variable - *vectors*. The *dMin* variable contains the number of the cluster that the current vector belongs to.

We decided to parallelize at the level of the training sample. This choice is due to the large number of input vectors. For parallelization, select the required number of blocks of 256 threads (to cover the number of input vectors). Each thread is responsible for a single vector in the training sample. Thus, we eliminated the loop on all images that significantly speeds up the algorithm. The Distance function is performed on the graphics device, and its code is shown in Fig. 2.24.

```
__device__ void Distance(int vectorNumber, float* x[],
float* w[], int* cl){
    float min = FLT_MAX, temp;
    int num = 0;
    for (int i = 0; i < CLUSTER; i++){
        temp = 0.0;
        for (int j = 0; j < N; j++){
            float r = w[i][j] - x[vectorNumber][j];
            temp += r * r;
        }
        if (temp < min){
            min = temp;
            num = i;
        }
    }
    *cl = num;
}
```

Fig. 2.24: Distance function that calculates the distance between vectors on a graphics device



Fig. 2.25 shows the function code on the GPU that trains the Kohonen network. Here, *devRate* denotes the  $\theta$  parameter. We can notice that in the *for* loop, changing the weights of *w* can be performed simultaneously by several threads. However, due to the large sample size and, consequently, frequent changes in weights, this "conflict" of writing new values to the array is not critical, and practically does not affect the result.

```

__global__ void TrainingParal(float* w[], float* x[],
float h){
    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    int dMin;
    if (tid < vectors)
        do{
            Distance(tid, pattern, w, &dMin);
            for (int i = 0; i < N; i++){
                w[dMin][i] += h * (x[tid][i] - w[dMin][i]);
            }
            __syncthreads();
            h *= devRate;
        } while (h > min_h);
}

```

Fig. 2.25: A core that represents a parallel implementation of network learning

Measure the learning time of Kohonen network on the CPU and GPU for each digit. Here is a histogram with the results obtained in Fig. 2.26.

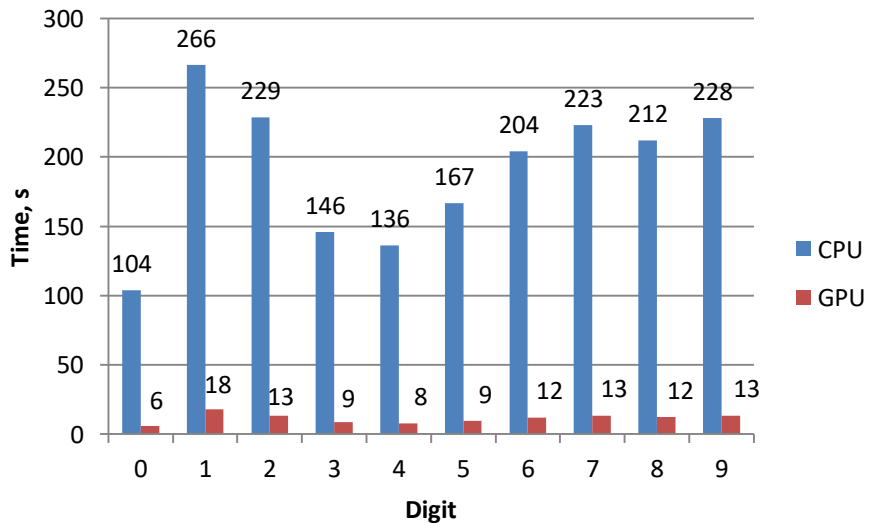


Fig. 2.26: Execution time of training of the Kohonen neural network on the CPU and on the GPU

Based on the algorithm's runtime data, you can calculate the acceleration obtained using CUDA technology for each digit. The acceleration coefficient is calculated as a simple ratio of the execution time of the algorithm on the CPU to the execution time of the algorithm on the GPU. The results are shown in Table 2.20.

Digit	0	1	2	3	4	5	6	7	8	9
Acceleration	17.2	14.8	17.4	16.9	17.5	17.5	17.2	16.5	17.2	17.0

Table 2.20: GPU acceleration

Thus, after analyzing the results shown in Fig. 2.24 and Table. 2.19, we can conclude that the learning of the Kohonen neural network is accelerated by an average of 16.9 times. Let us provide Table 2.21 with the characteristics of the computer where the calculations were performed (in Visual Studio 2019).

<b>OS</b>	Windows 10 Pro x64
<b>GPU model</b>	Nvidia GeForce GTX 1050TI
<b>GDDR capacity</b>	4 GB
<b>CPU model</b>	Intel Core i7-7700HQ 2.80 GHz
<b>RAM</b>	24 GB

Table 2.21: Computer characteristics

Let us perform parallelization using OpenMP and compare the results obtained on CUDA. For the accuracy of the results, we will also parallelize at the level of the training sample, meaning that each vector image will be executed on a different process.

First, let's compare the execution time of the algorithm using OpenMP technology and on a single CPU. The results are shown in Fig. 2.27.

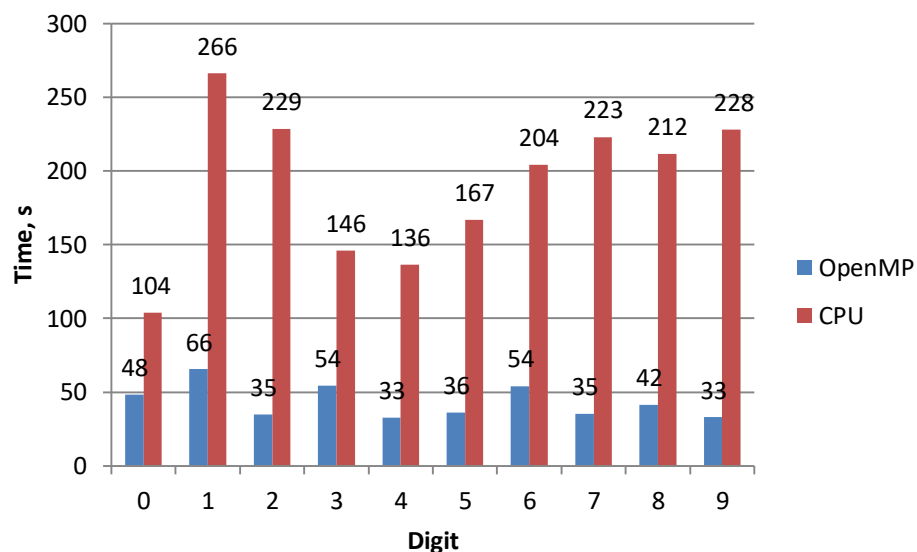


Fig. 2.27: Execution time of training of the Kohonen neural network on the CPU and on the OpenMP

Based on the algorithm's runtime data, you can calculate the acceleration obtained using OpenMP for each digit. The acceleration coefficient is calculated as a simple ratio of the algorithm execution time on the CPU to the algorithm execution time using OpenMP. The results are shown in Table 2.22.

<b>Digit</b>	0	1	2	3	4	5	6	7	8	9
<b>Acceleration</b>	2.2	4.1	6.6	2.7	4.2	4.6	3.8	6.3	5.1	6.9

Table 2.22: OpenMP acceleration

Thus, after analyzing the results shown in Fig. 2.25 and Table. 2.21, I can conclude that the learning of the Kohonen neural network is accelerated by an average of 4.6 times.

Let us analyze the parallelization performance on OpenMP using Intel Parallel Studio VTune. We will analyze all the numbers at once. So, the sequential algorithm runs for 1019 seconds. (Fig. 2.28)

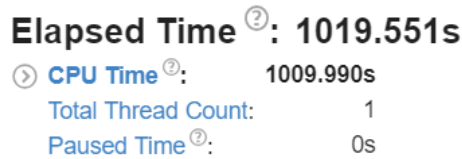


Fig. 2.28: Execution time of the Kohonen neural network training on the CPU for all digits

Now let us give a histogram of work on a single processor. This histogram shows the percentage of time that a certain number of processors were running simultaneously.

**Effective CPU Utilization Histogram**

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU utilization value.

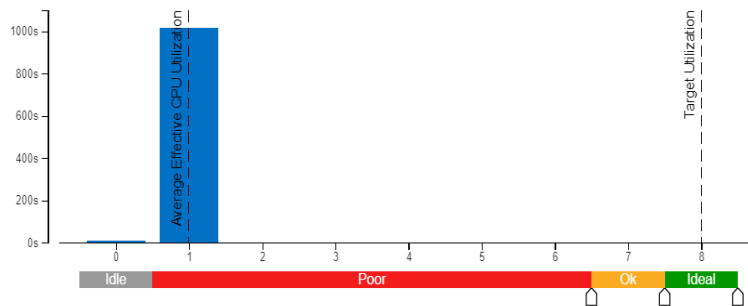


Fig. 2.29: Effective CPU utilization histogram

Similar actions are performed for code using parallelization.



Fig. 2.30: Execution time of the Kohonen neural network training for all digits using OpenMP

The execution time of the algorithm is 441 seconds that significantly improved the sequential result. Next, we will output a histogram of work on different numbers of processors.

Effective CPU Utilization Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU utilization value.

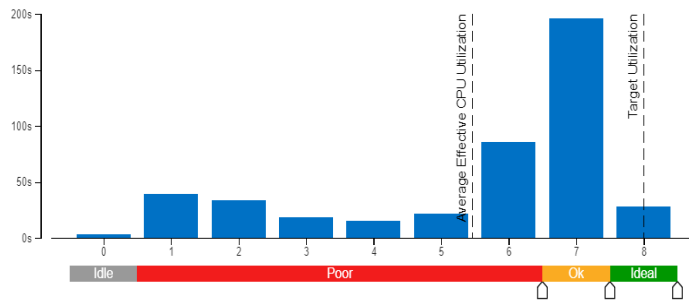


Fig. 2.31: Effective CPU utilization histogram

Analyzing Fig. 2.31, we can conclude that it is most efficient to use 4 logical processors. The operating time when using 4 processors is the least. Using 7 logical processors is not profitable in time aspect.

Thus, parallelization was performed using CUDA technology and using OpenMP. Let us compare the execution time of the algorithm on CUDA and on OpenMP.

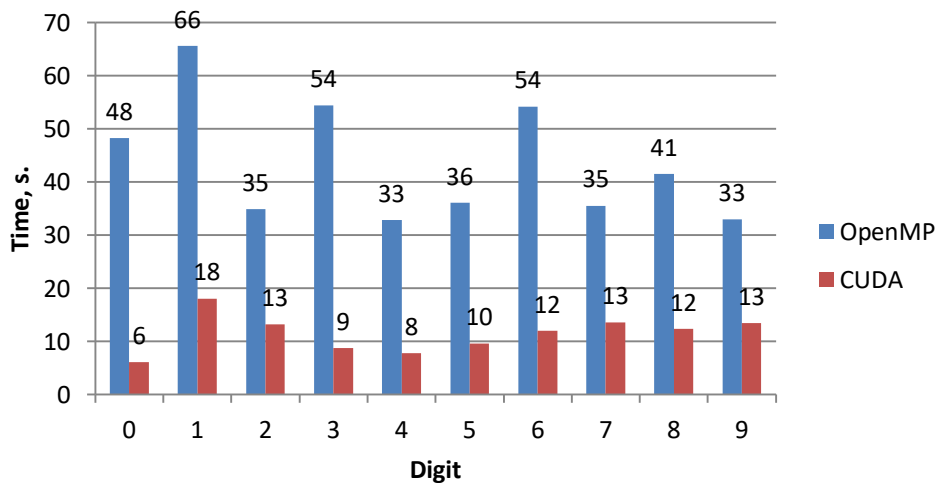


Fig. 2.32: The execution time of the Kohonen neural network training on the GPU and using OpenMP for each digit

Thus, according to the data obtained from Fig. 2.32, the CUDA technology gives significantly higher acceleration. Using OpenMP speeds up the learning algorithm of the Kohonen neural network by about 4.6 times, while parallelization using CUDA technology speeds up the algorithm by about 17 times. Thus, it is better to use CUDA technology for large-volume samples.

The last stage of recognizing images of handwritten numbers in the MNIST database is recognition by the Hopfield neural network. The architecture and algorithm were discussed in detail in the previous chapter.

# Chapter 3

## Testing

### 3.1 Testing on the Fashion MNIST dataset

In this work, it is necessary to show the universality of the neural networks used, that is, to demonstrate that this software can recognize not only handwritten numbers, but also other images. The Fashion MNIST dataset was selected for this purpose.

Just like the MNIST set, the Fashion MNIST set contains data measuring 28 by 28 pixels. The set contains ten types of clothing: t-shirt, shorts, sweater, dress, raincoat, sandals, shirt, sneakers, bag, shoes. There are 60,000 images in the training sample and 10,000 images in the test sample.

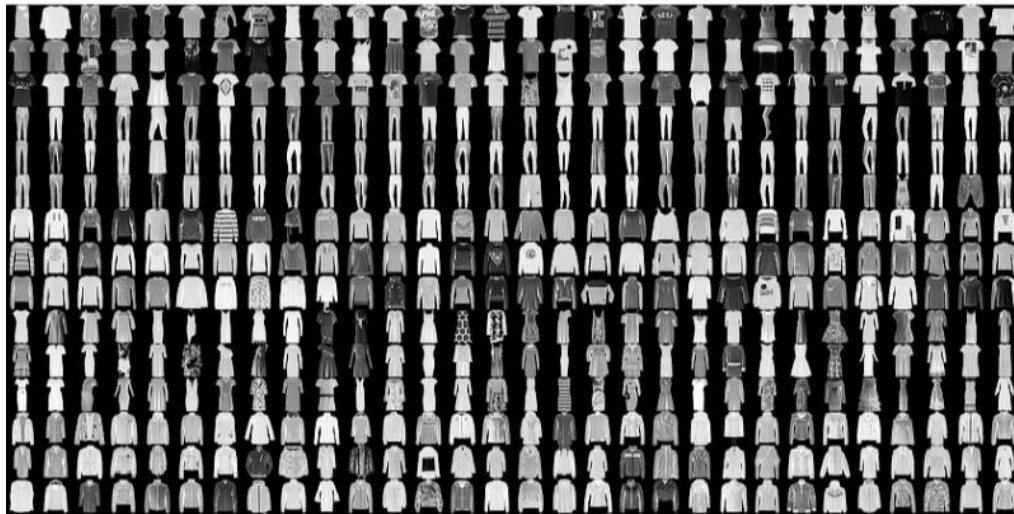


Fig. 3.1: Examples of images from the Fashion MNIST database

Let us determine the optimal number of clusters for this data set.

Class	t shirt	shorts	sweater	dress	raincoat	sandals	shirt	sneakers	bag	shoes
Clusters	25	37	29	37	44	46	13	49	1	40

Table 3.1: The optimal number of clusters for each image with a full sample

As can be seen from Table. 3.1 only one cluster was defined for bags. Now let us build clusters and give examples of cluster centers.

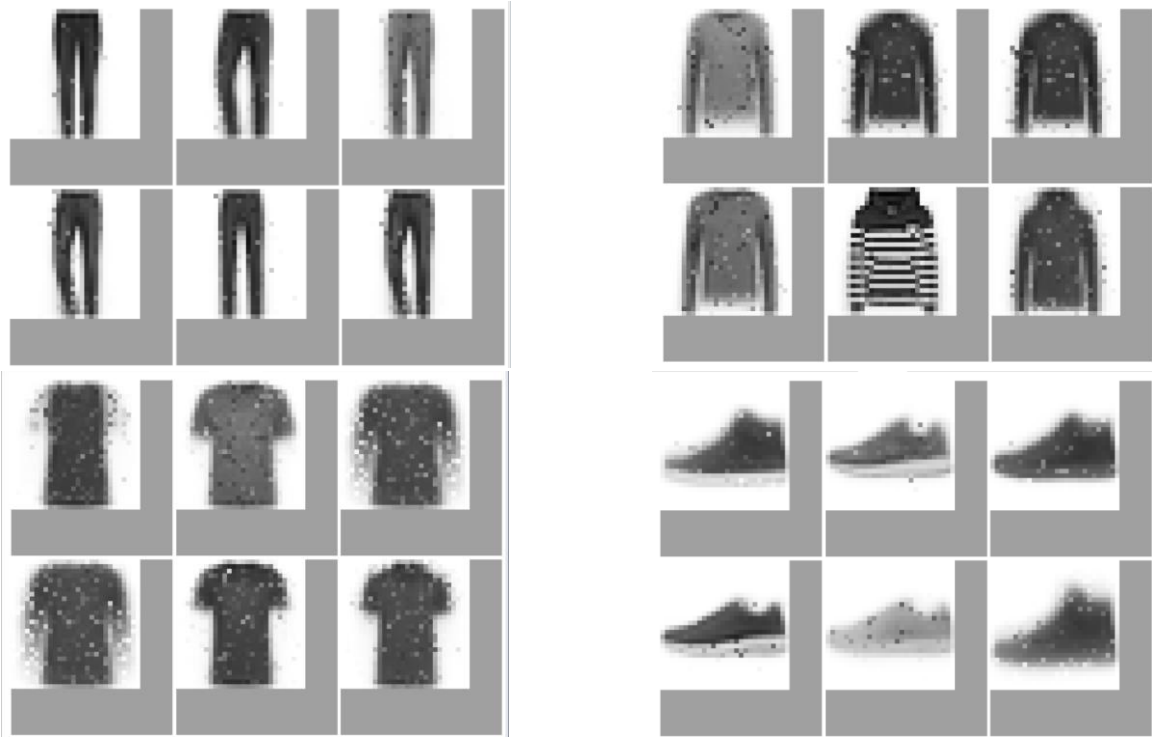


Table 3.2: Examples of cluster centers

Cluster centers are less clear for this set. Therefore, errors may occur when building clusters. This may be due to the number of clusters in the Table. 3.1 is not enough for this set.

Now we present a table of errors in recognition by the Kohonen neural network. For convenience, each class is marked with a number from 0 to 9 (t-shirt-0, shorts-1, and so on). Let's check the accuracy of cluster construction. To do this, for a test sample of the Fashion MNIST database, we will estimate the percentage of images contained in the desired cluster. To do this, measure the distance from each image to all the resulting clusters for each digit. If the image vector is closest to the cluster that belongs to the cluster group for the same number, then we assume that the vector is in the correct cluster. Fig. 3.2 shows the percentages of images from the test sample in clusters of their digits.

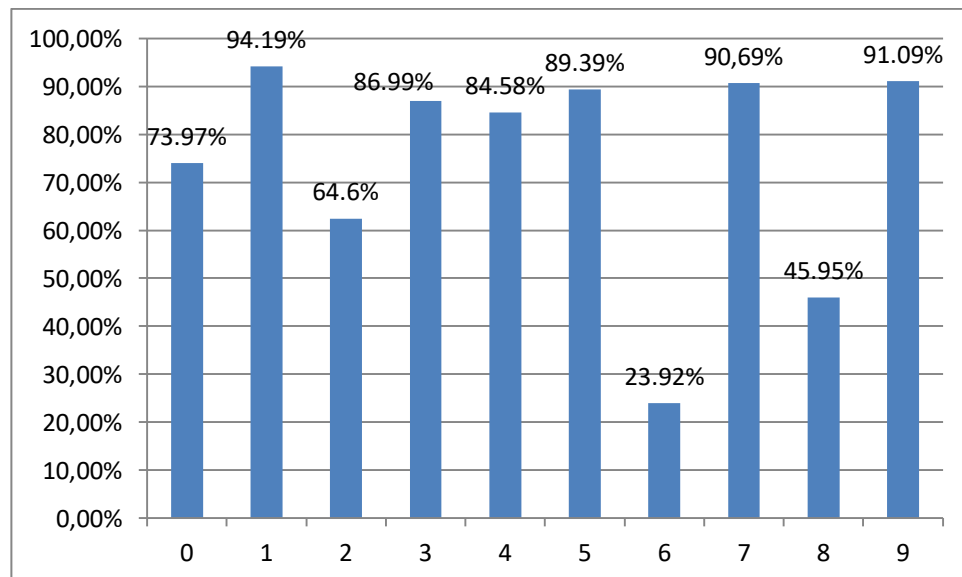


Fig. 3.2: Percentage of images from the test sample that are included in your cluster

As you can see from Fig. 3.2 class 6 - shirts are very poorly recognized, and t-shirts, sneakers and shoes are quite well. The average percentage of recognition by the Kohonen neural network is approximately 74 %. In Table 3.3 shows a summary table showing where the Kohonen neural network made errors.

	0	1	2	3	4	5	6	7	8	9
0	739	7	26	93	28	39	65	1	1	0
1	7	941	9	31	6	1	3	0	1	0
2	9	1	624	11	291	13	50	0	0	0
3	22	14	6	869	49	9	29	0	1	0
4	0	2	71	50	845	10	21	0	0	0
5	0	0	0	0	0	893	0	70	0	36
6	194	2	158	62	294	46	239	1	3	0
7	0	0	0	0	0	32	0	906	0	61
8	13	2	66	55	192	98	49	56	459	9
9	0	0	0	0	0	24	0	64	1	910

Table 3.3: Summary results of recognition by the Kohonen neural network

Also, to evaluate the accuracy of clustering of the Kohonen neural network, we calculate Recall, Precision, and F-measure. Thus, the table with the F-measure has the form:

Class	0	1	2	3	4	5	6	7	8	9
Recall	0,751	0,971	0,650	0,742	0,496	0,767	0,524	0,825	0,985	0,896
Precision	0,740	0,942	0,625	0,870	0,846	0,894	0,239	0,907	0,459	0,911
F-means	0,745	0,956	0,637	0,801	0,625	0,825	0,329	0,864	0,627	0,903

Table 3.4: F-measure

Based on the data described above, shirts are most often recognized as raincoats, this is due to the fact that they are not very similar to each other.

The last stage of software operation is recognition by the Hopfield neural network. Here are the results of the first method.

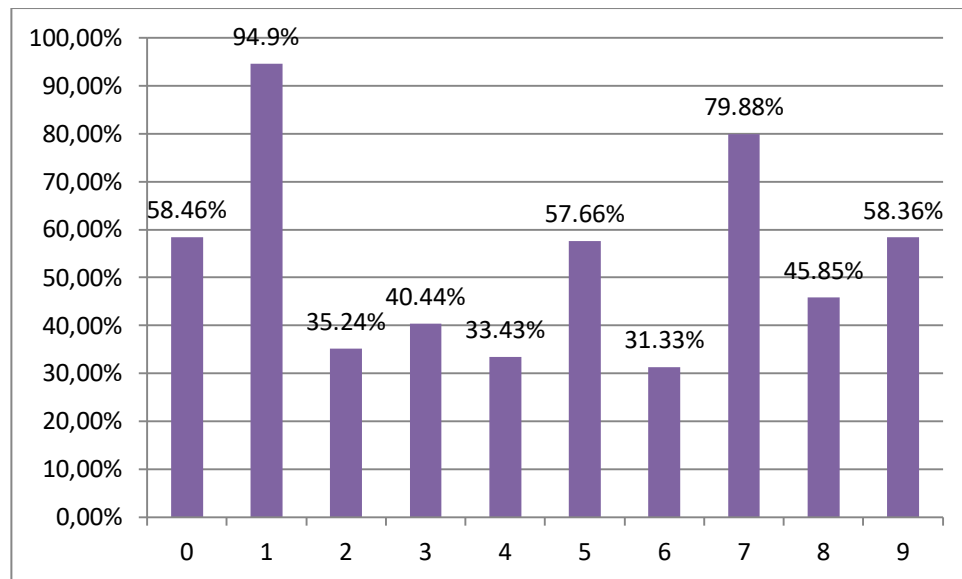


Fig. 3.3: Percentage of image recognition from the test sample by the Hopfield neural network method 1

As you can see from Fig. 3.3 class 6-shirts are also very poorly recognized, and t-shirts are quite good. The percentage of recognition for these categories is better than for recognition by the Kohonen neural network. The average percentage of recognition by the Hopfield neural network is approximately 53 %. This is because other categories are recognized worse. This is due to the network setting, and the network setting must be changed depending on the data. Table 3.5 shows a summary table showing where the Hopfield neural network made errors.

	0	1	2	3	4	5	6	7	8	9
0	584	44	14	128	2	8	96	99	9	15
1	3	945	1	30	4	0	9	6	1	0
2	10	36	352	33	127	28	242	137	18	16
3	16	464	0	404	9	0	36	55	3	12
4	4	23	163	129	334	5	265	59	8	9
5	0	29	0	5	0	576	13	317	0	59
6	132	32	88	141	84	26	313	133	17	33
7	0	7	0	0	0	168	12	798	0	14
8	1	21	60	105	33	20	26	209	458	66
9	0	0	0	1	0	40	0	374	1	583

Table 3.5: Summary results of recognition by the Hopfield neural network

Also, to evaluate the accuracy of clustering of the Hopfield neural network, we calculate Recall, Precision, and F-measure. Thus, the table with the F-measure has the form:

Class	0	1	2	3	4	5	6	7	8	9
Recall	0,779	0,590	0,519	0,414	0,563	0,661	0,309	0,365	0,889	0,722
Precision	0,585	0,946	0,352	0,404	0,334	0,577	0,313	0,799	0,458	0,584
F-means	0,668	0,727	0,420	0,409	0,420	0,616	0,311	0,501	0,605	0,646

Table 3.6: F-measure



Based on the data described above, we can draw the following conclusions. Images of sweaters can be confused with images of shirts, and sandals can be confused with sneakers. Initially, the database stores blurred images (fuzzy images), which also causes such large errors in speech recognition. Now we present the results of the second method.

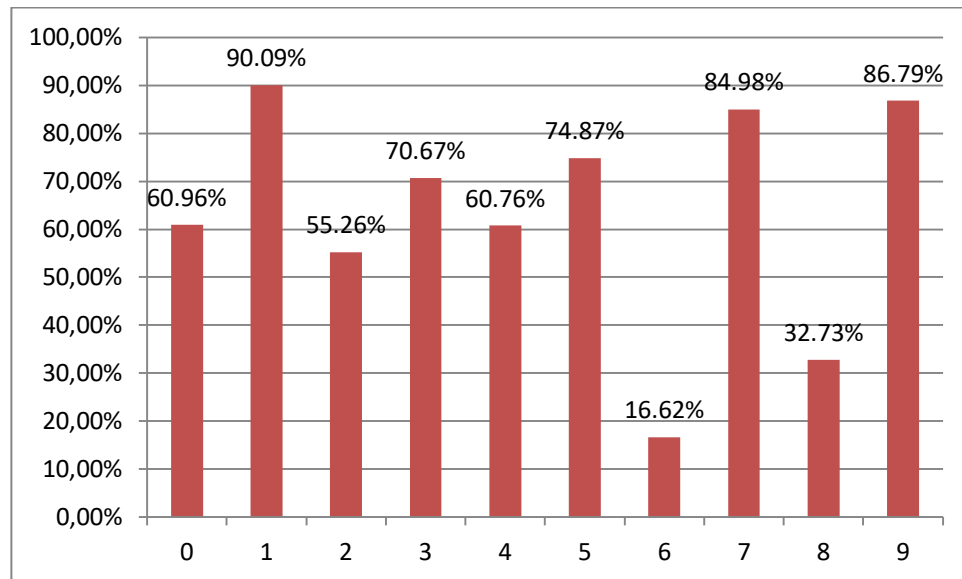


Fig. 3.4: Percentage of image recognition from the test sample by the Hopfield neural network method 2

As you can see from Fig. 3.4 class 6-shirts are also very poorly recognized, and t-shirts are quite good. The percentage of recognition for these categories is worse than for method 1 recognition. The average percentage of recognition by the Hopfield neural network using the second method is approximately 63 %. This result is better than using the first method. This is because other categories are recognized better. Table 3.7 shows a summary table showing where the Hopfield neural network made errors.

	0	1	2	3	4	5	6	7	8	9
0	609	36	71	87	29	82	66	14	2	3
1	11	900	8	65	3	3	9	0	0	0
2	38	24	552	52	196	59	66	3	3	6
3	50	126	20	706	38	34	17	1	0	7
4	20	15	182	96	607	28	45	1	1	4
5	9	3	16	4	2	748	2	176	0	39
6	177	39	240	78	176	113	166	1	3	6
7	0	0	0	0	0	87	0	849	0	63
8	20	6	251	30	74	73	7	109	327	102
9	2	1	1	14	5	42	1	66	0	867

Table 3.7: Summary results of recognition by the Hopfield neural network

Also, to evaluate the accuracy of clustering of the Hopfield neural network, we calculate Recall, Precision, and F-measure. Thus, the table with the F-measure has the form:

Class	0	1	2	3	4	5	6	7	8	9
-------	---	---	---	---	---	---	---	---	---	---

<b>Recall</b>	0,651	0,783	0,412	0,624	0,537	0,589	0,438	0,696	0,973	0,790
<b>Precision</b>	0,610	0,901	0,553	0,707	0,608	0,749	0,166	0,850	0,327	0,868
<b>F-means</b>	0,629	0,838	0,472	0,663	0,570	0,660	0,241	0,765	0,490	0,827

Table 3.8: F-measure

Based on the data described above, we can draw the following conclusions. Images of sweaters can be confused with images of raincoats, shirts can be confused with sweaters.

Here are examples of correct and incorrect operation of the program.


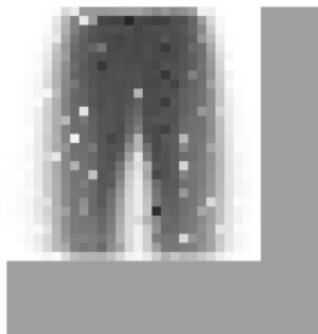


Image	Recognized as
	
	

Table 3.9: Examples

# Conclusion

In this work, we have studied the general information about neural networks used for pattern recognition, as well as written software for recognizing images of handwritten numbers. Software consists of two neural networks: the Kohonen neural network and the Hopfield neural network. The algorithms were tested on the basis of handwritten numbers - MNIST and on the basis of Fashion MNIST. Testing on different databases has shown the universality of the algorithms. A cluster analysis was also performed based on this database.

Clustering of handwritten numbers from the MNIST database using a Kohonen neural network. The optimal number of clusters is defined for each digit. The percentage of vectors from the test sample falling into the correct cluster for each digit is more than 90%. This means that clusterization can be used for pattern recognition of digits. The best clusterization is obtained for the digits 0 and 1 (a-measure is 0.974), the worst clusterization is for the digit 9, and its F-measure is 0.903.

Kohonen neural network training algorithm is parallelized on the GPU using CUDA technology. Parallelization is performed at the level of the training sample. The acceleration of the algorithm was approximately 17 times. The proposed approach can be used for clustering big data. Also, the training of the Kohonen neural network is parallelized using OpenMP. The acceleration of the algorithm in this case is an average of 5 times. The efficiency of parallelization on OpenMP on different number of processors was analyzed. It was concluded that the use of CUDA is more effective.

In this work, we propose a Hopfield neural network for recognizing handwritten numbers. Cluster centers constructed applying a Kohonen neural network served as objects to remember.

For the Hopfield neural network, two methods were studied and analyzed to compare 2 images. As a conclusion, we can say that the method 1 for recognizing images of handwritten numbers gives better results than method 2. This is because method 2 does not take into account that the image can be shifted: it can be positioned higher or lower, to the right or to the left. Also, when recognizing numbers, numbers often contain elements of each other. When testing algorithms on a dataset with images of clothing, the second method gave better results. The percentage of recognition (digits) when using the first method is 74%, and when using the second method is 64%.

The software was implemented in C#. The training of the Kohonen neural network and its parallelization was also performed in C++. This paper also shows the process of software development and describes the individual stages of work.

Further research suggests improving the software, as well as using it not only for images of handwritten numbers, but also generalize it to different types of images.

# Bibliography

- [1] **Zhang, X., Zhou, X., Lin, M., and Sun, J.** (2018) "ShuffleNet: An extremely efficient convolutional neural network for mobile devices." in Proceedings IEEE Conference on Computer Vision and Pattern Recognition: 6848–6856
- [2] **Dautov, R., and Mosin, S.** (2018) "Technique to aggregate classes of analog fault diagnostic data based on association rule mining." in Proceedings of 19th International Symposium on Quality Electronic Design: 238–243.
- [3] **Neha, S., Vibhor, J., and Anju, M.** (2018) "An analysis of convolutional neural networks for image classification." *Procedia Computer Science* 132: 377–384.
- [4] **Mesbah, A., Berrahou, A., Hammouchi, H., Berbia, H., and Daoudi, M.** (2019) "Lip reading with hahn convolutional neural networks." *Image and Vision Computing* 88: 76–83.
- [5] The MNIST database handwritten digits. URL: <http://yann.lecun.com/exdb/mnist>.
- [6] **Ramya, C., Kavitha, G, and Dr. Shreedhara, K., S.** "Recalling of images using Hopfield Neural Network Model." In National Conference on Computers, Communication and Controls – 11: N4C-11.
- [7] **Aksenov S.V., Novoseltsev V.B.** (2006) "Organization and use of neural networks (methods and technologies)." Tomsk: NTL.
- [8] **Lulu, C., Munggaran, Surryarini, Widodo, Cipta, A.M.** (2014) "Handwritten pattern recognition using Kohonen neural network based on pixel character." *International Journal of Advanced Computer Science and Applications* Vol.5: No. 11.
- [9] **Rexy M., Lavanya K.** (2019) "Handwritten digit recognition of MNIST data using consensus clustering." *International Journal of Recent Technology and Engineering*. Vol. 7: No. 6, P. 1969–1973.
- [10] **Schmidhuber, J.** (2015) "Deep learning in neural networks: An overview." *Neural Networks* 61: 85–117.
- [11] **Miri, E., Razavi, S.M., Sadri, J.** (2011): Performance optimization of neural networks in handwritten digit recognition using intelligent fuzzy c-means clustering. 1st Int. Conf. on Comp. and Knowledge Eng., 150-155.
- [12] **Pourmohammad, S., Soosahabi, R., Maida, A.S.** (2013): An efficient character recognition scheme based on k-means clustering. 5th Int. Conf. on Modeling, Simulation and Applied Opt., 1-6. <https://doi.org/10.1109/ICMSAO.2013.6552640>
- [13] **Li, B.Y.** (2018): An experiment of k-means initialization strategies on handwritten digits dataset. *Intelligent Inf. Management* (10), 43-48. <https://doi.org/10.4236/iim.2018.102003>

- [14] **Munggaran, L.C., Widodo, S., Cipta, A.M.** (2014): Handwritten pattern recognition using Ko-honen neural network based on pixel character. *Int. J. Adv. Comput. Sci. App.* 5(11), 1-6. <https://www.doi.org/10.5220/00066111002800360006611100280036>
- [15] **Senkovskaya I.,S., Saraev P.,V.** (2011) “ Automatic clustering in data analysis based on Kohonen self-organizing maps. ” *Bulletin of MSTU. Nosova I., G.:* No. 2, P.78-79.
- [16] Nordstrom, T.: *Designing parallel computers for self-organizing maps.* Linkoping, (1992).
- [17] **Shal, S.A., Koltun, V.** (2017): Robust continuous clustering. *P. Natl. Acad. Sci. USA* 114(37), 9814-9817. <https://doi.org/10.1073/pnas.1700770114>
- [18] **Jang, H., Park, A., Jung, K.** (2008): Neural network implementation using CUDA and Open MP *Proceedings of the International Conference on Digital Image Comput.: Techniques and App.* <https://doi.org/10.1109/DICTA.2008.82>
- [19] **Wittek, P., Daranyi, S.** (2012): A GPU-accelerated algorithm for self-organizing maps in a distributed environment. *European Symp. on Artificial Neural Networks, Comput. Intelligence and Machine Learning*, 609-614.
- [20] **Daneshpajouh, H., Delisle, P., Boisson, J.C., Krajecki, M., Zakaria, N.** (2018): Parallel batch Self-Organizing Map on graphics processing unit using CUDA. *High Perf. Comput.*, 87-100. [https://doi.org/10.1007/978-3-319-73353-1\\_6](https://doi.org/10.1007/978-3-319-73353-1_6)
- [21] **Liu, Y., Sun, J., Yao, Q., Wang, S., Zheng, K., Liu, Y.** (2018): A scalable heterogeneous parallel SOM based on MPI/CUDA. *Proc. of Machine Learning Research* (95), 264-279 (2018).
- [22] **McConnell, S., Sturgeon, R., Henry, G., Mayne, A., Hurley, R.** (2012): Scalability of Self-Organizing Maps on a GPU cluster using OpenCL and CUDA. *High Perf. Comput. Symp., J. Phys.: Conf. Ser.* (341), <https://doi.org/10.1088/1742-6596/341/1/012018>
- [23] **Takatsuka, M., Bui, M.** (2010): Parallel batch training of the Self-Organizing Map using OpenCL, pp. 470-476. Springer, Berlin, Heidelberg.
- [24] **Xu, Y., Zhang, W.** (2010): On a clustering method for handwritten digit recognition. *3rd Int. Conf. on Intelligent Net. and Intelligent Sys.* 112-115. <https://doi.org/10.1109/ICINIS.2010.130>