Ministry of Science and Higher Education of the Republic of Kazakhstan

Suleyman Demirel University

Alua Kusmanova

Text recognition using neural networks

THESIS

Presented in Partial Fulfilment for the

Master of Technical Sciences Degree in Computer Science

(degree code: 7M06102)

Department of Computer Science

Faculty of Engineering and Natural Sciences

Supervisor: **Associate Professor, Lyazzat Atymtayeva**

Kaskelen 2023

Suleyman Demirel University
Faculty of Engineering and Natural Sciences
Department of Computer Science

Dean of Faculty

Associate Professor

PhD Zhamanov A.

06 06 2023

Topic of the thesis:

Text recognition using neural networks

Thesis submitted as part of the requirements for the award of the MSc in "7M06102 - Computer Science" SDU, 2021-2023

Head of Department _____ Assistant Professor, PhD Mukash Zh.

Academic Supervisor _____ Associate Professor, Atymtayeva L.

Master student _____ Kusmanova Alua

Kaskelen 2023

Declaration

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Alua Kusmanova

2023

Acknowledgements

In order to complete this job, I must recognise and extend my sincere gratitude to my supervisor, assoc.prof. Atymtayeva L. B. I was able to complete all of my project's writing stages because to her direction and assistance. Additionally, I want to thank the members of my committee for making my defense a fun experience and for their insightful comments and ideas.

Additionally, I want to express my gratitude to my entire family for their unwavering support and patience as I conducted my research and wrote my project. Your supplication for me has kept me going so far.

Finally, I want to express my gratitude to God for guiding me through all of the challenges. Every day, I have felt your guiding.

Dedication

I dedicate this dissertation to my parents, for your constant love and support.

Abstract

Text recognition is one of the key tasks in the field of deep learning. In recent years, neural networks and deep learning have attracted increasing attention in the context of developing effective and accurate models for text recognition. This scientific dissertation explores the use of neural networks and the Python programming language to solve the problem of text recognition.

First, an overview of existing methods and approaches to text recognition, including classical rule-based methods and traditional machine learning, is presented.

The following is an implementation of text recognition using the TensorFlow library and the Python programming language. The principles of data preprocessing, including tokenization, vectorization and normalization, are described. Experiments are being conducted on various data sets, including handwritten text and text on images. Various hyperparameters of the model are investigated, such as the network architecture, the size of the hidden layer and the learning rate.

Thus, this scientific dissertation confirms the importance and effectiveness of using neural networks and the Python programming language for the task of text recognition. This opens up prospects for further research and development in the field of text recognition and other natural language processing tasks.

Аңдатпа

Мәтінді тану - терең оқыту саласындағы негізгі міндеттердің бірі. Соңғы жылдары нейрондық желілер мен терең оқыту мәтінді танудың тиімді және дәл үлгілерін әзірлеу контекстінде көбірек назар аударуда. Бұл ғылыми диссертация мәтінді тану мәселесін шешу үшін нейрондық желілерді және Python бағдарламалау тілін пайдалануды зерттейді.

Біріншіден, мәтінді танудың қолданыстағы әдістері мен тәсілдеріне, соның ішінде классикалық ережелерге негізделген әдістерге және дәстүрлі машиналық оқытуға шолу жасалады және нейрондық желілердің негізгі принциптерін қарастырылады.

Төменде TensorFlow кітапханасы мен Python бағдарламалау тілін қолдана отырып мәтінді тануды жүзеге асыру берілген. Әр түрлі мәліметтер жиынтығымен, соның ішінде қолжазба мәтінімен және суреттердегі мәтінмен эксперименттер жүргізілуде. Модельдің әртүрлі гиперпараметрлері зерттеледі, мысалы, желі архитектурасы, жасырын қабат өлшемі және оқу жылдамдығы.

Осылайша, бұл ғылыми диссертация мәтінді тану мәселесін шешу үшін нейрондық желілер мен Python бағдарламалау тілін қолданудың маңыздылығын растайды. Бұл мәтінді тану және табиғи тілді өңдеудің басқа міндеттері бойынша қосымша зерттеулер мен әзірлемелерге мүмкіндіктерді ашады.

Аннотация

Одной из основных задач глубокого обучения является распознавание текста. Нейронные сети и глубокое обучение становятся все более востребованными в процессе создания эффективных и точных моделей распознавания текста. Эта научная диссертация рассматривает, как решить проблему распознавания текста с помощью нейронных сетей и языка программирования Python.

Во-первых, рассматриваются различные подходы к распознаванию текста, включая классические правила и традиционное машинное обучение. Далее представлена пример использования распознавания текста с использованием языка программирования Python и библиотеки TensorFlow.

Результаты эксперимента подтверждают точность и эффективность нейронных сетей в решении проблемы распознавания текста. Модели были разработаны и оптимизированы с использованием языка программирования Python и библиотеки TensorFlow для высокой точности распознавания текста в различных сценариях.

Таким образом, эта научная диссертация подтверждает, что использование нейронных сетей и языка программирования Python для решения проблемы распознавания текста является как полезным, так и эффективным. Это открывает новые возможности для исследований и разработок в области распознавания текста и других областей обработки естественного языка.

Abbreviations

NN - Neural Networks

CNN - Convolutional Neural Network

RNN - Recurrent Neural Networks

AI - Artificial Intelligence

DL - Deep Learning

ML - Macine Learning

OCR - Optical Character Recognition

CV - Computer Vision

NLP - Natural Language Processing

Table of Contents

Chapter 1

Background and motivations

1.1 Introduction

Currently, computerization in our society is growing very rapidly and plays a very significant role in human daily activities. With the help of computer technologies, attempts are now being made to automate a wide range of processes that in the recent past were attributed to human mental abilities, since information technologies are used everywhere, and therefore programmers around the world are developing new and improving algorithms for automating routine processes that have already been implemented and proven in practice.

One of the urgent problems at the moment is the inefficient work of various state organizations, such as city polyclinics, various organizations, since everything is done manually. Each form is completed by hand, and the operator himself enters any information (such as passport information) that has to be added to the database. Almost all governmental and commercial entities today need to automate the process of converting information from paper documents to electronic format. The solution to this issue is crucial for security and control as well because even the smallest data entry error can have serious consequences. Utilizing software that enables automation of this process can help you avoid this. It can be used in various areas of work related to document processing. An illustration of the above is the practical activity of companies that register personal data of

citizens: utilities, passport offices, accounting at the place of work, etc. Modern information developments allow us to significantly facilitate access to the necessary information resources from anywhere in the world, provided their electronic version is available.

Currently, there are quite a few optical character recognition systems, but, unfortunately, not all of them are truly high-quality products. So, "low-level" text recognition programs translate text on paper into a set of characters, and then offer to figure out what happened on their own. Systems with high speed and accuracy of recognition are usually very expensive, which makes them practically inaccessible for implementation and practical use on a mass scale.

The object of research in this case is character recognition algorithms, and the subject of research is an optical character recognition system based on images.

Objective: to develop an algorithm of increased accuracy for optical character recognition from images by combining preprocessing methods using the Tensorflow text recognition library and dictionary control of the results obtained [1].

1.2 Aims and Objectives

One of the most important and challenging tasks in CV and NLP processing right now is handwriting recognition. There are several uses for handwriting recognition, including automating data entry into computers and searching through handwritten documents, as well as enabling a number of applications in the fields of medical, banking, archival storage, and other industries.

The purpose of this dissertation is to develop and research effective methods and algorithms for handwriting recognition. The main goal of the work is to increase the accuracy and speed of handwriting recognition, as well as to reduce recognition errors.

It is necessary to complete the following tasks in order to reach this goal:

- Study of handwriting recognition techniques currently in use. This includes the analysis of modern algorithms and models based on artificial neural

networks, machine learning and statistical methods. You should also become familiar with the different data sets that are used to train and test handwriting recognition models.

- Creation of novel handwriting recognition techniques and algorithms. It is required to create new models and algorithms that take into consideration the unique characteristics of handwriting and have high recognition accuracy and speed based on an analysis of existing approaches.
- Implementation and experimental study of the developed methods. The developed methods and algorithms should be implemented in the form of software and subjected to experimental testing. As part of the experiments, it is necessary to conduct a comparative analysis of the results of handwriting recognition using various approaches and models.
- Evaluation of the effectiveness of the developed methods and algorithms. After conducting the experiments, it is necessary to analyze the results obtained and evaluate the effectiveness of the developed methods and algorithms in comparison with existing approaches. This includes evaluating the accuracy, speed, and stability of handwriting recognition.

When recognizing handwritten text using recurrent neural networks, RNN models can be used to analyze the sequence of characters in the input data and predict the corresponding text. LSTM networks, which are a kind of RNN, have the ability to preserve long-term dependencies between characters in a sequence and are widely used for processing data sequences, such as handwritten text [2].

Chapter 2

Neural Network: what is it and its classification

2.1 Neural networks: general information

In theory, artificial neural networks have been studied and used since the turn of the 20th century, but it wasn't until much later that they gained significant public recognition. This is mainly because technologically sophisticated (for the time) computing processors started to emerge, whose capacities were sufficient to support artificial neural networks. In reality, a medium complexity neural network can currently be easily simulated on any desktop computer. A group of neurons that are specifically interconnected make up a neural network. Think about one neuron(see Figure 2.1):

A neuron is a component that uses a number of input signals to calculate an output signal (in accordance with a predetermined rule). One neuron's primary chain of events is comprised of the following specific actions:

- obtaining signals from prior network components; combining input signals;
- calculating output signals;
- transmitting the resultant signal to subsequent neural network components.

The structure of a given network determines the many diverse ways that neu-

rons can connect to one another. The neural network's core, however, never changes. One or more output signals are created based on all of the signals that are input to the network. In other words, a neural network might be reduced to a basic "black box" having inputs and outputs. Additionally, this compartment contains a substantial number of neurons.

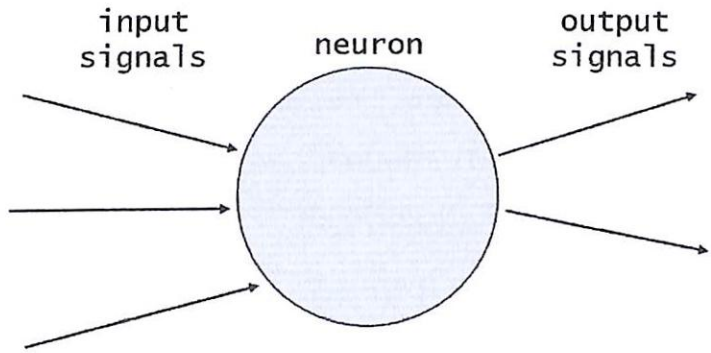

Figure 2.1: Structure of a simple neuron

Let's now concentrate on each stage of this network individually as we have listed its major phases.

A precise rule for integrating all of these inputs must be devised when designing a neural network because each neuron can receive several input signals. And it is widely used to summarize the weighted values of relationships. Why is anything "weighted"? Let's solve it now.

Every link in a network of neurons may be accurately described by three characteristics:

- the element from which it originates,
- the element to which it is directed,
- the weight of the connexion [3].

The third factor currently holds our greater attention. Depending on the link's weight, the signal sent over it will either be amplified or attenuated. Let's take a look at this example if you want to put it simply(see Figure 2.2):

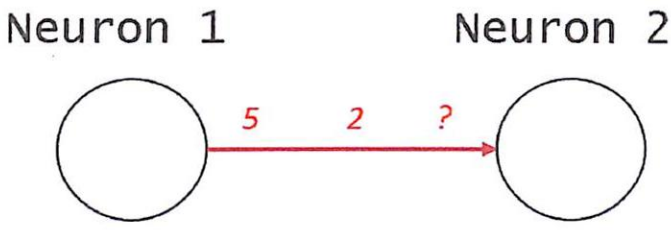

Figure 2.2: Example to the simple neuron

Neuron 1's output signal is 5. The neuronal connection has a weight of 2. To calculate the input signals of neuron 2 coming from neuron 1, multiply the value of this signal by the weight of the connection (5 * 2).

If there are multiple signals, every one of them add up as well. We notice the following at the neuron's input as a result:

$$net_i = \sum_{i=1}^{n} x_i * w_{ij} \tag{2.1.1}$$

The term net_j in this formula refers to the outcome of adding up all of the input signals for neuron j. There are N elements sending signal j's input with their output signals. And w_{ij} displays the strength of the link between neurons i and j. By summing up every of the weighting input signals, we can determine the network element's combined input. Most frequently, a matrix W, also known as a weight matrix, is used to illustrate the topology of connections between neurons. The statement states that the element of the matrix w_ij determines the amount of weight of the relationship between element i and element j. Let's look at a basic neural network to better understand how weight matrices are created(see Figure 2.3):

Such a neural network's weight matrix will take the following format:

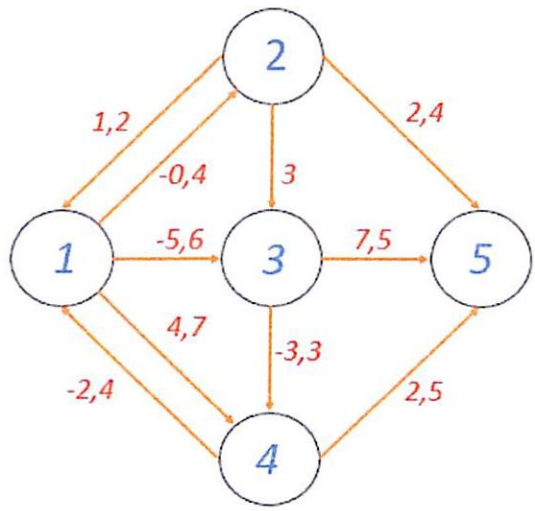

Figure 2.3: Schema of the simple neural network

$$W = \begin{bmatrix} 0 & -0.4 & -5.6 & 4.7 & 0 \\ 1.2 & 0 & 3 & 0 & 2.4 \\ 0 & 0 & 0 & -3.3 & 7.5 \\ -2.4 & 0 & 0 & 0 & 2.5 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.1.2}$$

For instance, a connection with a weight of three exists between the second and third elements. Look at the matrix; the number 3 in the second row and third column indicates that everything is in order [4].

2.2 Function for element activation

Take a look at the output signals. Based on a particular principle, the value of each network element's output is determined by adding the values of all of its input elements. The element activation function is the name of this regulation. And neuron activity is the term used for the output value itself. Any mathematical function, let alone some of the more popular ones, can work as an activation function.

- logistic function;

- threshold function - if the total input value falls below a predetermined threshold, the activity is zero; if it rises, the activity is one.

Let's examine another brief example that is commonly used in research to highlight the essential principles underlying neural networks' operation. The objective of the example is to use a neural network to calculate the XOR ratio. To put it another way, we will offer a variety of signal input options, and our output should include the results of an XOR operation given the input values(see Figure 2.4):

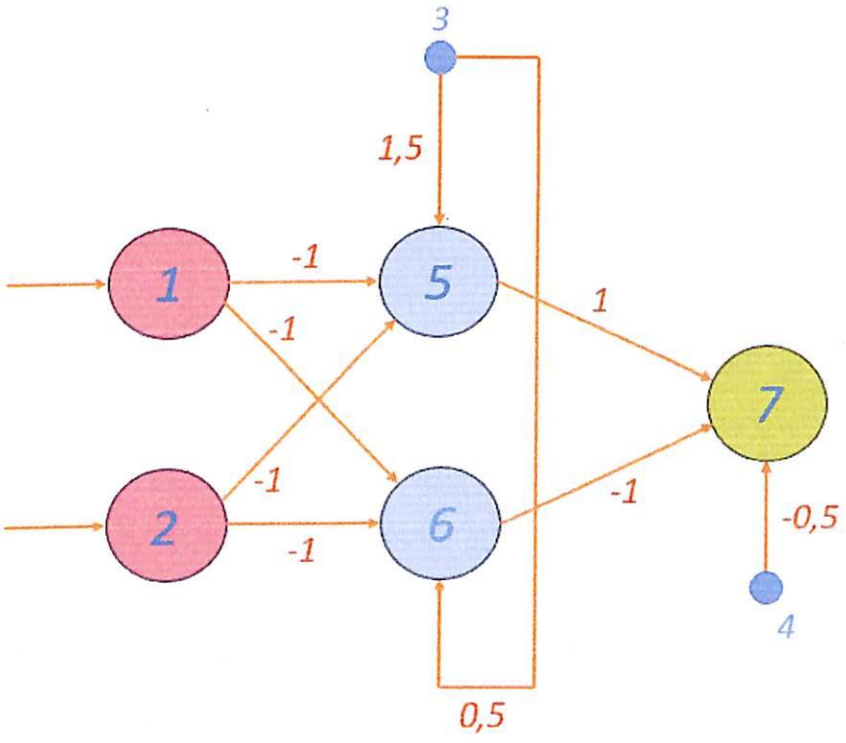

Figure 2.4: Example of neural network to determine the XOR ratio

Element 7 is the output, and elements 1 and 2 are the inputs. Because they have no connection to the outside world, neural networks 5 and 6 are known as hidden neurons. There are consequently three types of layers: input, concealment, then output. The terms "offset elements" refer to elements 3 and 4. Their signal of output (activity) is consistently 1. The threshold function will serve as an activity function, and the rule of summation of weighted connections will be used to determine the total input in this network. An element's activity is equal to zero if all of its inputs together are less than zero; if they are more than zero, the activity is equal to one. Give input to neuron 1 a value of one, and input to

neuron 2 a value of zero. In this situation, the output should result in 1 (0 XOR 1 = 1).

The fundamental feature of neural networks is their capacity to independently adjust the weight values of all connections during network training, even though in this case we were aware of the values of all the weighting coefficients in advance.

2.3 Classification of NN

Neural networks can be categorized according to the type of learning they use into two categories:

- supervised learning NN,
- unsupervised learning NN.

NN used supervised learning. When learning with a teacher, the target vector which represents the desired outcome, is presupposed to exist for each input vector. A training pair is the name given to them collectively. On a given number of these training pairings, the network is typically trained. After computing the network output and comparing it to the associated target vector, the output vector was displayed. The weights also change due to an algorithm that strives to reduce the error. The weights are changed for each of the training set's sequentially given vectors as soon as the overall training array's error falls to a manageable level.

NN using unsupervised learning. Unsupervised learning is a much more logical learning approach in terms of the biological basis of artificial neural networks. Input vectors are the only objects in the training set. In order to produce consistent output vectors, that is, the same outputs when sufficiently identical input vectors are presented, the training phase alters the network's weights. As a result, the learning process draws attention to the training set's statistical characteristics and classifies related vectors into groups.

Neural networks can be categorized according to the setting weights they use into two groups:

- networks with dynamic connections, where synaptic weights are altered as

the neural network learns,

- networks with stable links, where the neural network's weights are instantly determined according to the details of the problem.

Neural networks are categorized by the sort of input information they receive into two groups:

- binary NN, in which all entered information is represented as zeros and ones, and
- analog NN, in which entered information is given as real numbers [5].

Chapter 3

Architectures of neural networks

There is a huge variety of different architectures, the main ones we will consider. But keep in mind that there are a lot of them. Let's look at several popular neural network architectures:

3.1 Single-layer neural network

One neuron can conduct the simplest calculations, but connections between neurons enable a neural network to accomplish its essential tasks. A single-layer perceptron is a straightforward neural network made up of several neurons arranged in a layer. A vector of values is used to encode the input data, and each element is supplied to the associated input of a different neuron in the layer. The neurons individually calculate the output after that. The output's dimension (or number of components) is equal to the number of neurons, and the number of synapses connecting each neuron should be the same across the board and match the size of the input signal(see Figure 3.1). Here X_1, X_2, X_3 is the input pattern, Y_1, Y_2, Y_3 is the output pattern, and $w_(i, j)$ is the j-th weighting factor of the i-th neuron.

Single-layer NN training

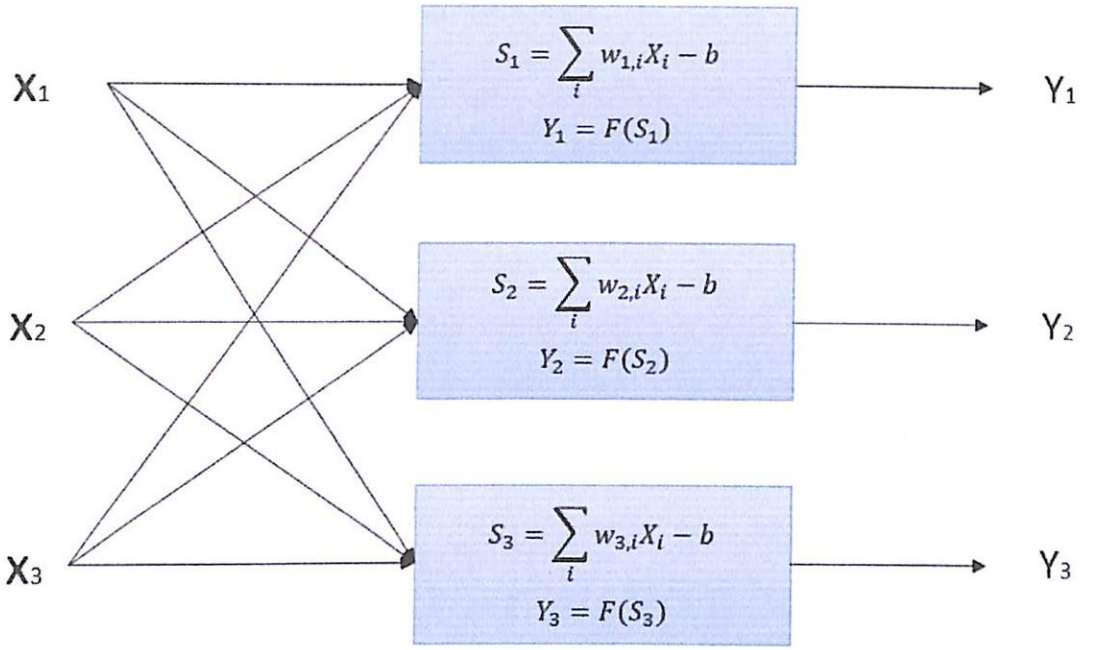

Figure 3.1: Single-layer neural network

The settings of a neural network are set up during the training phase by replicating the conditions in which the network is located. The type of training is impacted by how the settings are altered. There are algorithms for teaching both with and without a teacher. As part of the learning process with an instructor, a network of model training instances is offered. Every sample is processed after being sent to the neural network's inputs. After that, the output signal of the network is determined, and its value is compared to the target vector's equivalent value, which stands in for the desired network output.

The neural network must be trained in order to be able to do the task. For educating both with and without a teacher, there are algorithms. A network of model training examples is presented as part of the learning process with an instructor. Each sample is sent into the neural network's inputs, where it is processed. After that, the output signal of the network is determined, and its value is compared to the target vector's equivalent value, which stands in for the desired network output. The weighting parameters of the links within the network change based on the chosen method once the error is evaluated in accordance with a preset rule. The vectors in the training set are presented in order, errors are identified, and weights are changed for every vector until the error in the entire

training array is within acceptable bounds.

The training set solely contains input vectors when instructing without a teacher. In order to produce consistent output vectors—that is, the same outputs when sufficiently identical input vectors are presented—the training phase alters the network's weights. As a result, the learning process draws attention to the training set's statistical characteristics and classifies related vectors into groups. Even though presenting an input vector from this class will produce a certain output vector, it is challenging to forecast which output will be produced by this class of input vectors before training. As a result, the learning process ought to transform the network's outputs into something understandable. This is not a big issue. The network often makes it simple to identify the link between the input and output. The signal training approaches of Hebb and Oya are applied to train neural networks without a teacher.

The learning process can be conceptualized mathematically as follows. The neural network implements a function known as Y = G(X) and produces an output signal Y as it operates. If the network design is known, the synaptic weights and delay network values determine the type of function G.

Let's say the answer to the issue is a function $Y = F(X)$, where $(X_1, Y_1), (X_2, Y_2), \ldots$ are the input-output data parameters and $Yk = F(Xk)$, where (k = 1, 2, ..., N).

Finding (synthesizing) a function G that is similar to F in the sense of some error function E constitutes the training.

The training of a neural network becomes a multidimensional optimization problem with a very large dimension if a set of training examples is chosen, pairs (X_N, Y_N) (where k = 1, 2,..., N), and a method for calculating the error function E. This is because the error function E can have any form of training in the general case, which results in a multiextremal nonconvex optimization problem.

This issue can be resolved using the following (iterative) algorithms:

1. Local optimization techniques that compute first-order partial derivatives

   - the conjugate gradient method, which considers the direction of the antigradient at various algorithmic steps;

- the gradient algorithm (steepest descent method);

- methods with one- and two-dimensional optimization of the objective function in the direction of the anti-gradient;

2. Local optimization procedures that compute partial derivatives of the first and second orders, such as the:

   - Gauss-Newton algorithm,

   - the Levenberg-Marquardt algorithm,

   - optimization techniques with sparse Hesse matrices;

3. stochastic optimization algorithms, including the:

   - Monte Carlo method (a numerical method of statistical tests),

   - simulated annealing,

   - search in a random direction;

4. Algorithms for global optimization (global optimization issues are resolved by repeatedly changing the values of variables that affect the objective function)

3.2 Multi-layer NN

A multilayer NN (perceptron) has input, output, and one or more hidden layers of neurons arranged between them(see Figure 3.2).

The following algorithm must be used to choose the parameters of a multilayer perceptron:

- Ascertain the significance assigned to the X input vector's component parts. The formalized problem condition, or all the data required to obtain a response, must be included in the input vector.

- Choose the output vector Y so that each of its elements contains the entire solution to the problem.

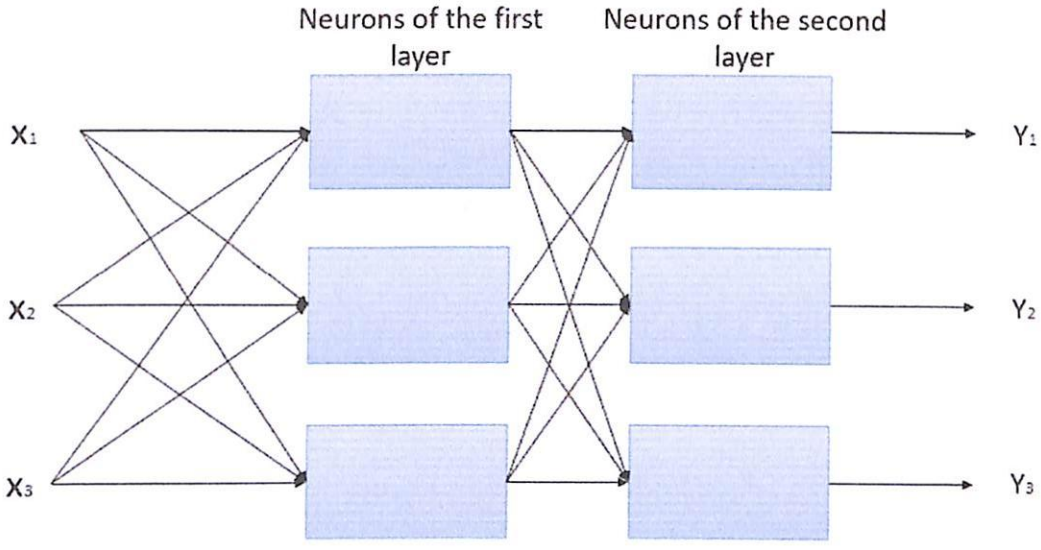

Figure 3.2: Multi-layer neural network

- Decide on the neuron activation function type. At the same time, it is preferable to consider the details of the assignment because a wise decision will quicken learning.
- Decide how many layers and neurons each layer will include.
- Decide on the activation function to use and then set the range of inputs, outputs, weights, and threshold levels accordingly.
- Give weights and threshold levels their original values. The horizontal section of the activation function of the neurons should not be saturated by the starting values; if it is, learning will proceed very slowly. The initial values shouldn't be too low, as this would cause the majority of the neurons' outputs to be equal to zero, which would slow down learning.
- Perform training, that is, choose the network parameters to enable the most effective resolution of the problem. The network will be able to solve the kinds of tasks it has been trained to solve once the training is complete.
- Enter the problem's conditions as a vector X in the network input. Determine the output vector Y to determine the problem's formalized solution.

3.3 Feedback networks

Information from later layers is communicated to earlier ones in networks with feedbacks. The following categories of neural networks include feedbacks:

1. layered-cyclic. All layers are equal and have the ability to both take input signals and make output, making them layered-cyclic, which is defined by the fact that the layers are closed in a ring and the last layer communicates its output signals to the first;

2. layered-fully connected networks, here signals are transported both between levels and inside layers in layered-completely connected networks, which are made up of layers, each of which is a fully linked network; Each layer's work cycle is broken down into three stages: signal reception from the layer below, signal exchange within the layer, signal production, and signal transmission to the layer above;

3. totally connected-layered, This is structurally similar to layered-entirely connected, but differs in that impulses are received by neurons in both the layer before and layer after at each cycle. They do not distinguish between the exchange phases inside the layer and the transmission phases.

The picture below shows the Elman and Jordan networks (see Figure 3.3):

By different neuronal structural types, neural networks can be classified as homogeneous (homogeneous) or heterogeneous. A heterogeneous network consists of neurons with various activation functions, whereas homogeneous networks only contain neurons of the same type.

Neural networks are also separated into synchronous and asynchronous categories. In the first case, each neuron modifies its state one at a time, whereas in the second case, the state of a large number of neurons, often throughout the entire layer, changes quickly. By repeatedly performing the same kinds of operations on neurons, algorithms are used to calculate how much time passes in neural networks [6].

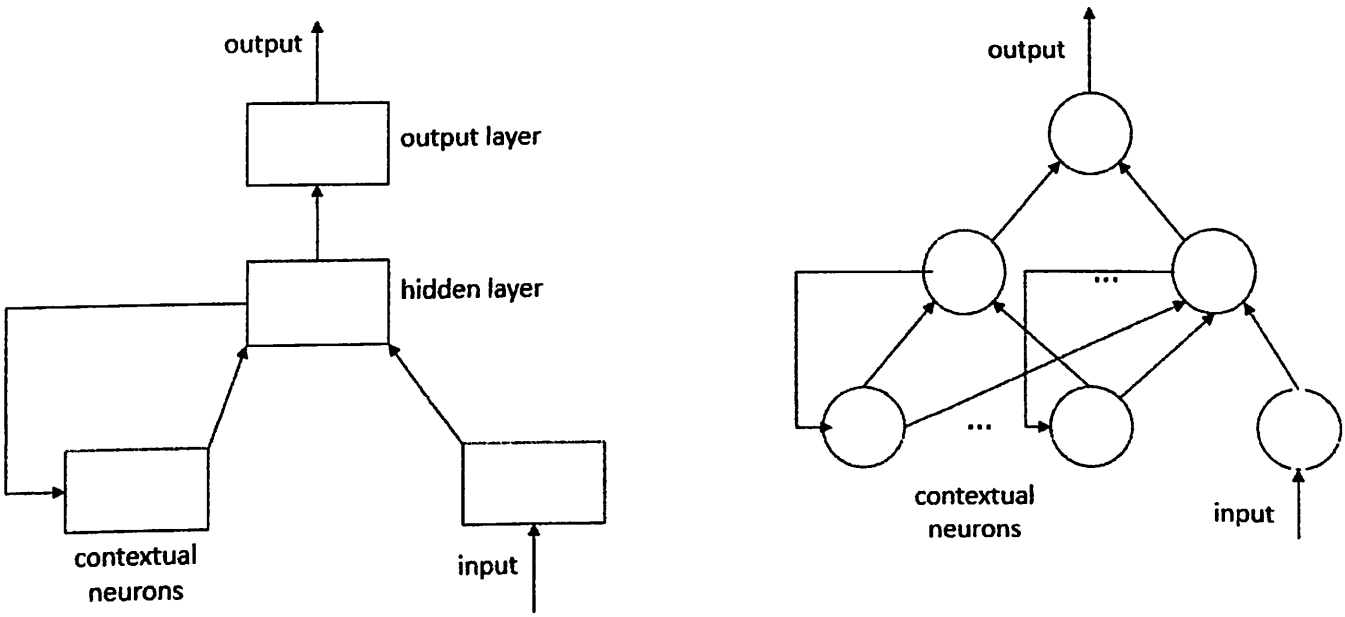

Figure 3.3: The Elman and Jordan networks

3.4 Formal neuron

A neuron only has one output channel and numerous input channels. The neuron receives task information through its input channels, forming the output outcomes of the work. A provided nonlinear function is used by the neuron to convert the outcome of computing the weighted sum of the input signals. The parameters of a neuron are a set that includes the threshold level and all weights.

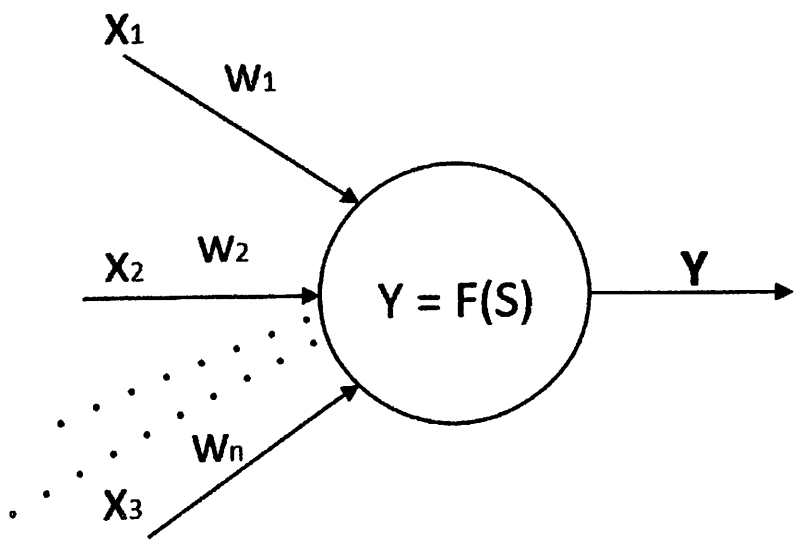

Figure 3.4: Structure and parameters of the neuron

The following designations are introduced here: $X_1, X_2, ..., X_n$ - input signal (pattern), $w_1, w_2, ..., w_n$ – weight coefficients, b - neuron threshold. The neuron first computes a weighted sum, $S = \sum_i w_i X_i - b$, next, the output signal Y is calculated using the activation function F(S).

The neuron activation process is a mechanism that controls a neuron's signal output. The total of all the signal weights and signals' products is the input for this function.

Let's examine the activation procedures that are most frequently employed:

- Threshold operation. This piecewise linear function is straightforward.
- Linear threshold. This piecewise linear function is straightforward. It consists of a segment where the function increases strictly monotonically.
- The sigmoid function (sigmoid). This nonlinear S-shaped function has a monotonically growing differentiable form. With sigmoid, you may enhance weak signals without becoming saturated by powerful signals.
- Hyperbolic Tangent (tanh). This method returns a real number in the range of 1 to 1 and accepts any real number as an input. The hyperbolic tangents can get saturated, much like a sigmoid. The output of this function is centered with respect to zero, unlike the sigmoid.

Disadvantages of the formal neuron:

- It is assumed that the neuron instantly calculates its output, therefore, with the help of such neurons, it is impossible to directly model systems with an internal state.
- Formal neurons, unlike biological ones, cannot process information synchronously.
- There are no clear algorithms for selecting the activation function.
- It is impossible to regulate the operation of the entire network.
- Excessive formalization of the concepts of "threshold" and "weight coefficients". In real neurons, the threshold changes dynamically, depending on

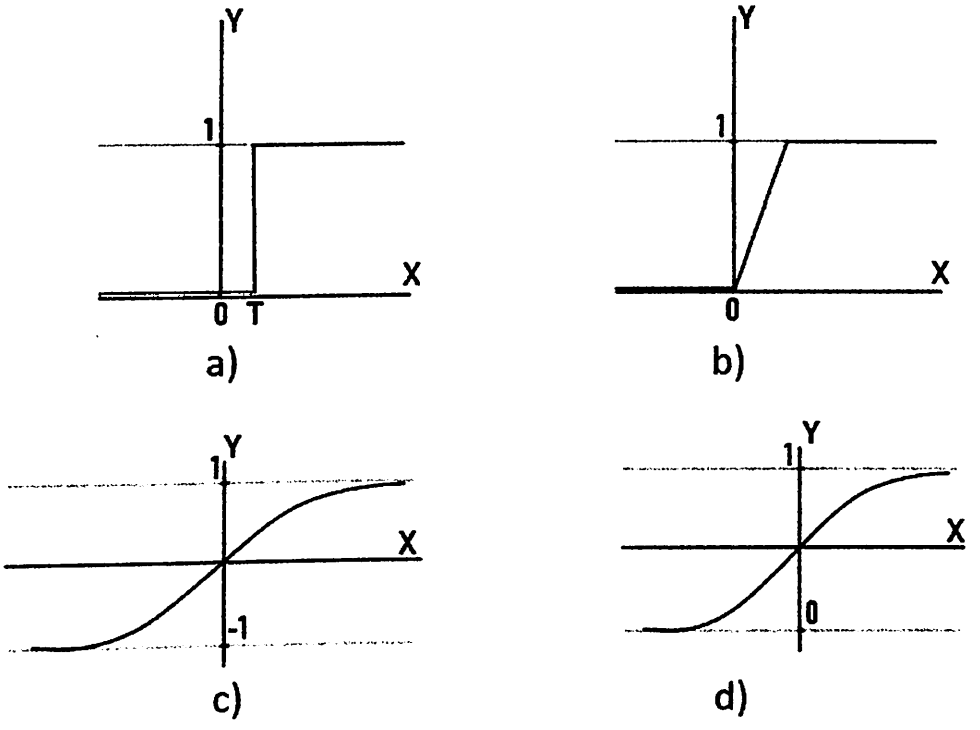

Figure 3.5: Neuron activation function

the activity of the neuron and the general state of the network, and the weight coefficients change depending on the passing signals.

3.5 Recurrent neural networks

Recurrent neural networks (RNN) are a highly reliable and promising type of neural network due to their unique ability to maintain internal memory. Being the only algorithm with this capability, RNNs excel in forecasting future events by leveraging their internal memory to retain important information about the input data they receive. This makes them particularly well-suited for handling sequential data such as time series, speech, text, financial data, audio, video, weather, and more. Compared to other algorithms, recurrent neural networks possess a deeper understanding of the sequence and its context.

The key characteristic of recurrent neural networks lies in their cyclic repetition of information. When making decisions, RNNs take into consideration not only the current input data but also the knowledge acquired from previous inputs. This short-term memory feature is intrinsic to recurrent neural networks. Moreover,

RNNs can further enhance their memory capabilities by incorporating long-term memory, often achieved through specialized units like Long Short-Term Memory (LSTM).

Recurrent neural networks (RNNs) differ from other algorithms in that they utilize both recent past and current inputs. This distinction is significant because RNNs can perform tasks that other algorithms struggle with, as the sequence of data holds crucial information about future events. Like other deep learning algorithms, a direct-coupled neural network assigns a weight matrix to each input before generating output data. However, in the case of RNNs, both the current input and the previous input receive weights, highlighting their interdependence. Moreover, RNNs adapt the weights over time through gradient descent and back-propagation. Unlike direct-coupled neural networks, RNNs have the capability to map one-to-many, many-to-many (e.g., translation), and many-to-one (e.g., speech classification), expanding their range of applications beyond one-to-one mapping.

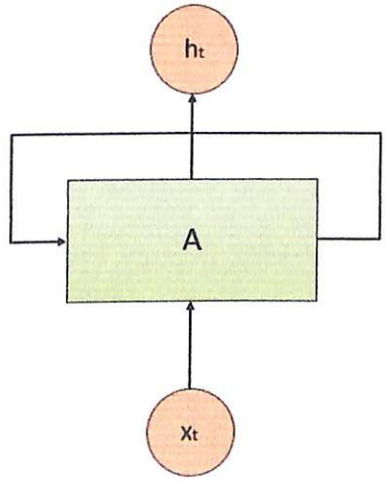

Figure 3.6: Reccurent Neural Networks

Recurrent neural networks contain feedbacks.

In the depicted diagram, fragment A of the neural network receives an input value x_t and produces an output value h_t. The inclusion of feedback in this architecture enables the transfer of information between different steps of the network, as illustrated in Figure 3.6.

The presence of feedback in recurrent neural networks often evokes a sense of intrigue. However, upon closer examination, they are not fundamentally distinct from conventional neural networks. A recurrent network can be conceptualized as multiple replicas of the same network, with each replica transmitting information to the subsequent replica. Here's what occurs when feedback is employed:

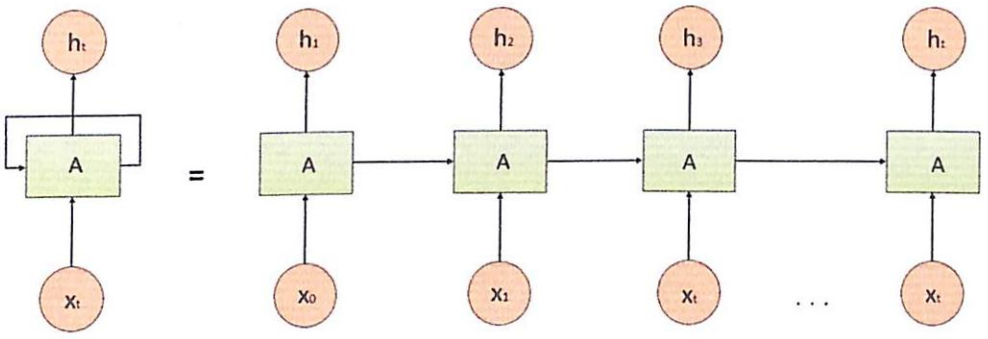

Figure 3.7: Recurrent neural network in the scan

The fact that RNNs resemble a chain suggests that they are closely related to sequences and lists. RNN is the most natural neural network architecture for working with this type of data [7].

The issue of long-term dependencies

One of the appealing aspects of recurrent neural networks (RNNs) is their potential ability to connect past information with the current task. For instance, having knowledge about the previous frame of a video could aid in understanding the current frame. If RNNs possess this capability, they would be highly valuable. However, the extent to which RNNs offer such an opportunity depends on certain factors.

In certain cases, we may only require recent information to fulfill the current task at hand. Consider a language model attempting to predict the next word based on previous words. When predicting the final word in the sentence "we love it," we don't necessarily need a broader context; it is evident that the last word will be "it." In situations where the gap between the relevant information and its application is small, RNNs can effectively learn how to utilize past information (see Figure 3.8).

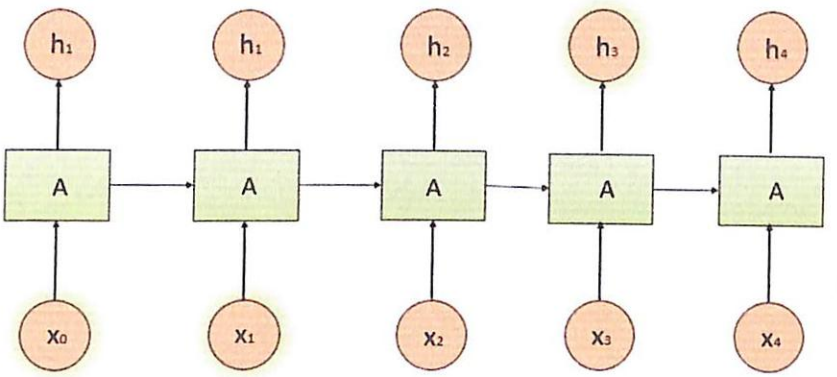

Figure 3.8: Recurrent neural network in process

LSTM Networks

Long short-term memory (LSTM) is a specialized architecture within the realm of recurrent neural networks that excels at learning long-term dependencies. First introduced by Sepp Hochreiter and Jurgen Schmidhuber in 1997, LSTM has since been further refined and widely adopted by numerous researchers.

LSTMs offer an effective solution to address the challenge of long-term dependencies. Unlike other architectures, LSTMs possess inherent capabilities to retain and utilize information over extended periods, without encountering difficulties in learning.

In general, recurrent neural networks can be visualized as a series of interconnected neural network modules. In a conventional RNN, each module typically comprises a simple structure, such as a single layer with the hyperbolic tangent (tanh) activation function [8].

3.6 Convolutional neural networks

Convolutional neural networks (CNN) are a subset of neural networks that are specifically designed to process input having a grid architecture, like images. Digital images are visual data that are represented in binary. It is based on consideration , justification in the form of grids and various grid cells, visual data such as brightness and color.

Convolutional layer, union layer, and fully linked layer are the standard three

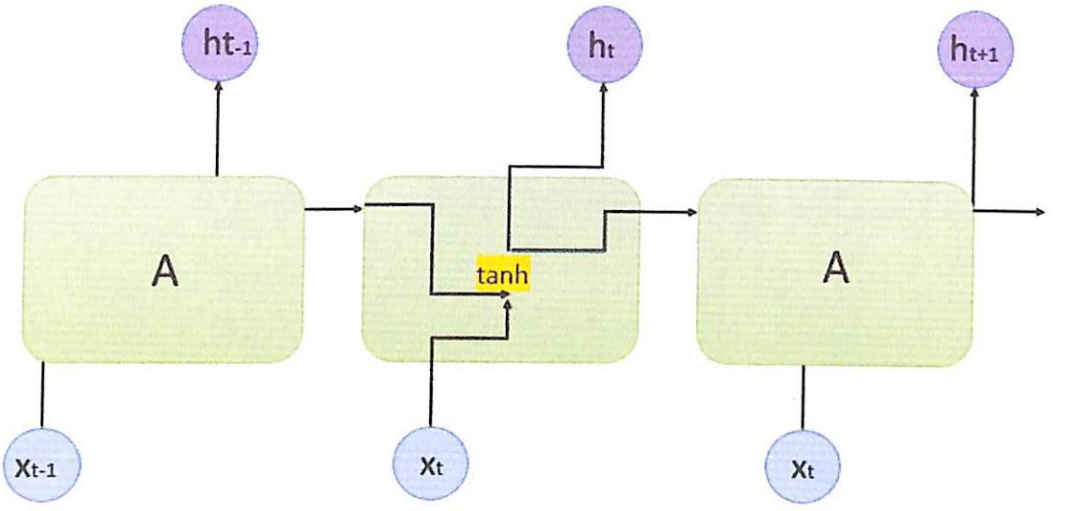

Figure 3.9: LSTM Networks

layers of CNN.(see Figure 3.10):

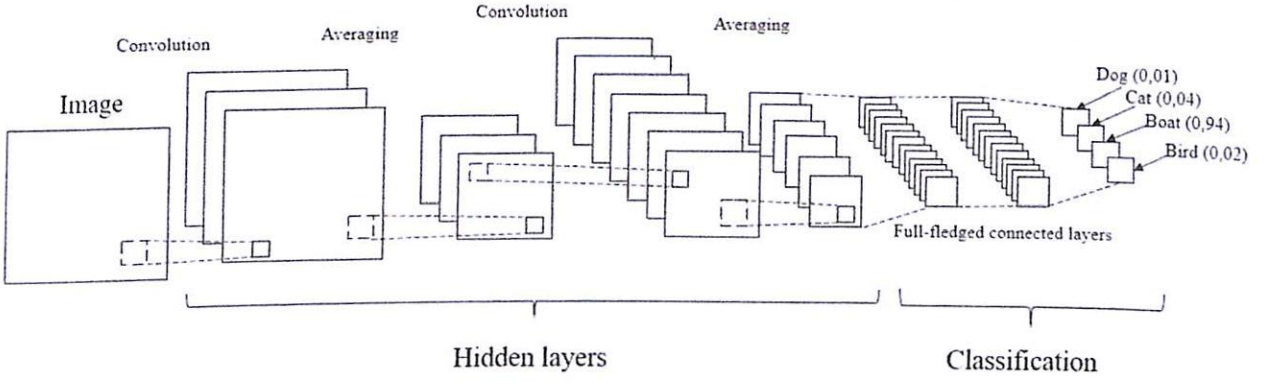

Figure 3.10: Architecture of Convolutional Neural Networks

Convolutional layer

The core element of CNN is the convolutional layer. On the network, it is primarily in charge of the compute load.

Between these two matrices, there is a scalar product. A limited area of the perceiving field is represented by one matrix, which is a set of common parameters utilized as a core. The outcome is very significant despite having a limited region. This means that if an image has three RGB channels, the depth of each channel expands while the width and depth of the core are relatively apparent.(see Figure 3.11):

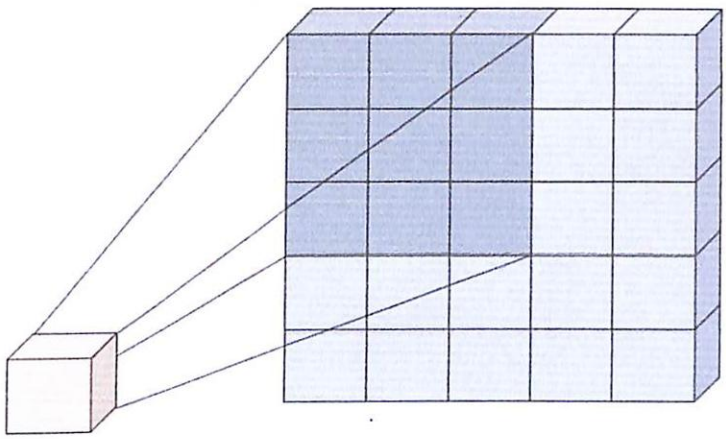

Figure 3.11: Convolution Process

The core creates an image representation of this receptive area by sliding along the height and width of the image during a forward pass. This creates a two-dimensional representation of the image, called an activation map, which allows the core to react to each position of the image. The decision step to change the size of the kernel [9].

If we have input data of size W x W x D, as well as the number of cores with an open space F with a step S and a fast retreat P , then the size of the output layer can be determined by the formula:

$$W_{out} = \frac{W - F + 2P}{S} \tag{3.6.1}$$

here, W_{out} is output layer size, W - input layer size, F - spatial size, P - number of margins, S - step.

We will get the output resolution : W_{out} x W_{out} x D_{out}. For more information, see Figure 3.12.

Reasons for convolution

Convolution leverages three fundamental concepts—sparse interaction, parameter sharing, and equivariant representation—which have greatly influenced researchers in the field of computer vision. Each of these concepts will be thoroughly explored and explained in the following sections.

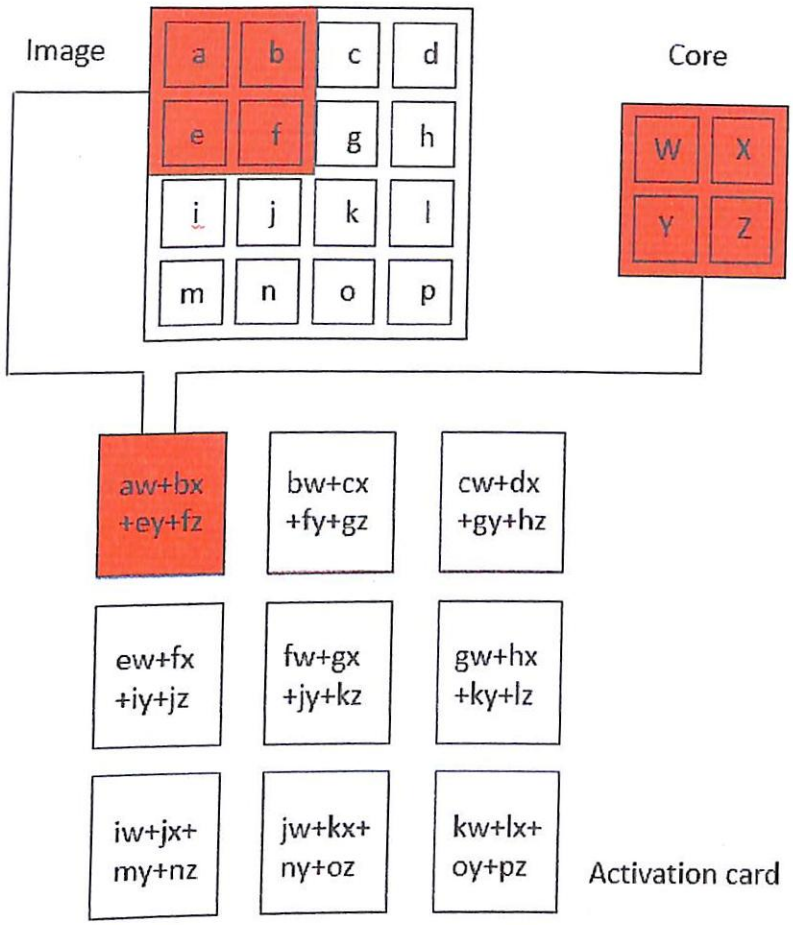

Figure 3.12: Convolution Operation

At the core of a neural network, the interaction between input and output is characterized by the multiplication of a parameter matrix. However, in convolutional neural networks, this interaction occurs selectively and sparsely. For example, when processing an image using a kernel, relevant information can be extracted from a subset of pixels, which is significantly smaller than the total number of pixels in the image. By reducing the size of the kernel relative to the input data, the number of parameters needed is also reduced, resulting in lower memory requirements for the model while enhancing its statistical effectiveness.

If calculating a specific feature at one spatial position (x1, y1) is valuable, it stands to reason that calculating a similar feature at another spatial position (x2, y2) would also be beneficial. This implies that neurons should utilize the same set of weights for each two-dimensional slice or generate a single activation map. By employing shared parameters, a convolutional network ensures that the weights applied to one input are identical to those applied to another. In contrast, in a

standard neural network, each element of the weight matrix is used only once and remains unchanged.

Furthermore, convolutional neural network layers exhibit translation equivalence, meaning that any alteration made to the input data will result in a corresponding change in the output. This property is achieved through the combined utilization of parameters and is a significant advantage of convolutional networks. [10].

Pooling Layer

In some locations, the pooling layer takes the place of the network output while obtaining summary statistics from the closest outputs. As a result, fewer calculations and weights are needed and the spatial dimension of the representation is reduced. Each view fragment has its own distinct processing for the merge procedure.

Various aggregation functions can be used to combine information from a neighborhood in image processing tasks. Some examples include calculating the mean of a rectangular neighborhood, computing the L2 norm for a rectangular neighborhood, or taking a weighted average based on the distance from the center pixel. However, the most prevalent approach is to identify the maximum deviation within the neighborhood, which is known as the maximum union.(see Figure 3.13).

To determine the size of the output volume, the following formula can be employed when given an activation map of dimensions W x W x D, a spatial size F kernel, and a step size S.

$$W_{out} = \frac{W - F + 2P}{S} + 1 \tag{3.6.2}$$

here, W_{out} is output layer size, W - input layer size, F - spatial size, S - step.

This will determine the output volume, which will have the dimensions W_{out}, W_{out}, and D_{out}.

Since the pooling always offers some degree of translation invariance, the item

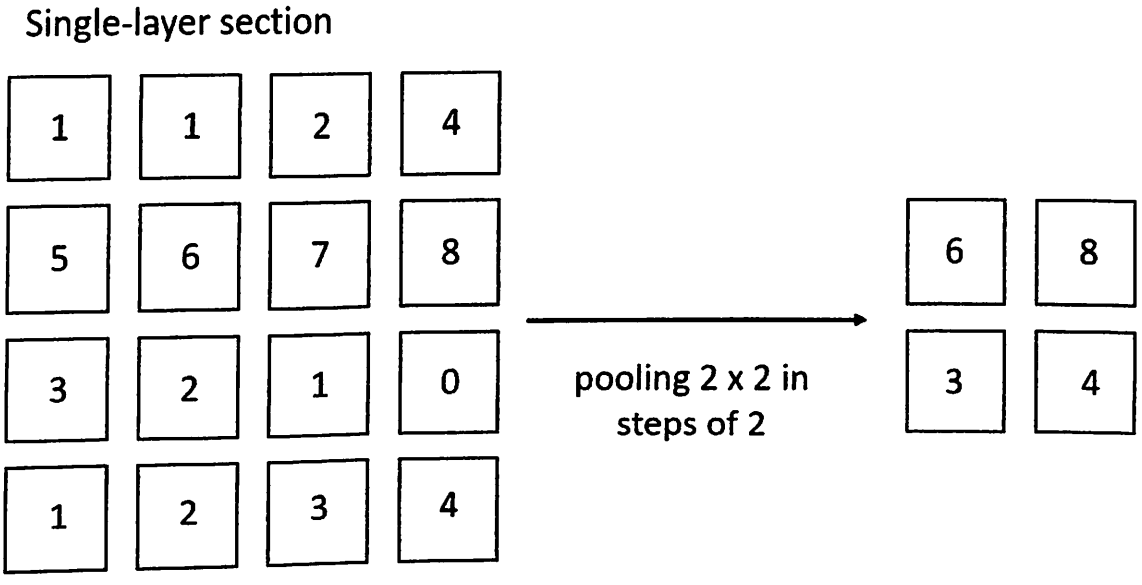

Figure 3.13: Convolution Operation

will always be recognisable regardless of where it is in the frame.

In the next chapter, we will understand in more detail how text recognition occurs and consider the methods of text recognition [11].

Chapter 4

Deep learning and its principles

4.1 What is deep learning?

Deep learning is widely used in the modern world, in anything from manufacturing to healthcare. Businesses are using this technology to address complicated issues including speech and object recognition, machine translation, and other related issues.

Although deep learning is obviously still far from perfect, it is already on the verge of providing economic advantages. For instance, these autonomous vehicles. Well-known businesses like Google, Tesla, and Uber are already attempting to bring autonomous vehicles to city streets. By 2021, Ford believes that the proportion of autonomous vehicles will have significantly increased. Additionally, the US government was able to create a set of security guidelines for them [12].

The Python-based high-level neural network API Keras can be used with TensorFlow, CNTK, or Teano. It was created with a focus on the potential for quick experimentation. The secret to conducting effective research is being able to move quickly from concept to conclusion.

You must comprehend how it works with artificial intelligence, neural networks,

and machine learning in order to respond to this query. To do this, we employ the concentric-circle visualization technique: (see Figure 4.1)

The outer circle represents all forms of artificial intelligence, such as computers. Deep learning and artificial neural networks are in the middle, machine learning is a little to the side.

Deep learning is essentially just a more accessible moniker for artificial neural networks. The term "deep" in this context refers to the neural network's level of complexity, which is frequently relatively shallow.

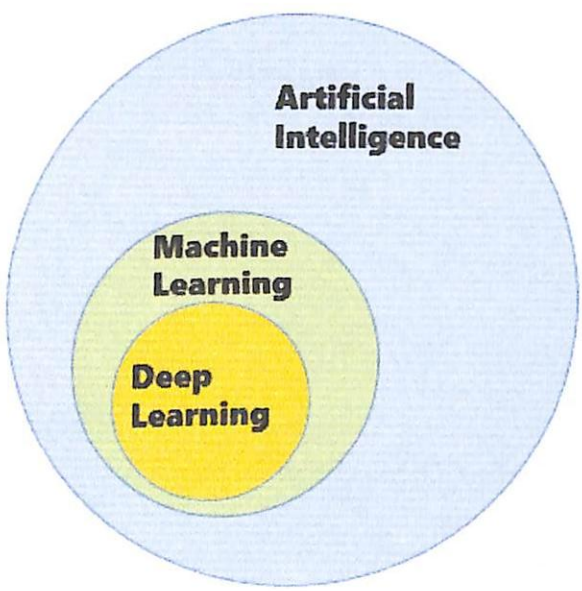

Figure 4.1: Concentric-circle visualization technique

The cerebral cortex's structure served as inspiration for the first neural network's designers. The perceptron, the network's fundamental building block, is essentially a mathematical equivalent of a biological neuron. Additionally, the neural network can exhibit perceptrons overlapping with one another, just like in the brain [13].

The input layer is the top layer of a neural network. A little amount of information is fed into each node in this layer before being transmitted to succeeding nodes in other layers. The final node in the chain produces the neural network's output when there are often no connections between the nodes of one layer.

Because they do not have connections to the outside world like the output and

input nodes, the middle nodes are known as hidden nodes. Only if the preceding levels are activated are they called (see Figure 4.2).

Deep learning is essentially an approach to neural network learning that makes use of numerous layers to solve complex tasks (like speech recognition) using templates. Because of the high cost and constrained data capacity in the eighties, the majority of neural networks were single-layer.

Deep learning is a particular kind of machine learning, if we think of it as an offshoot or a subset of artificial intelligence.

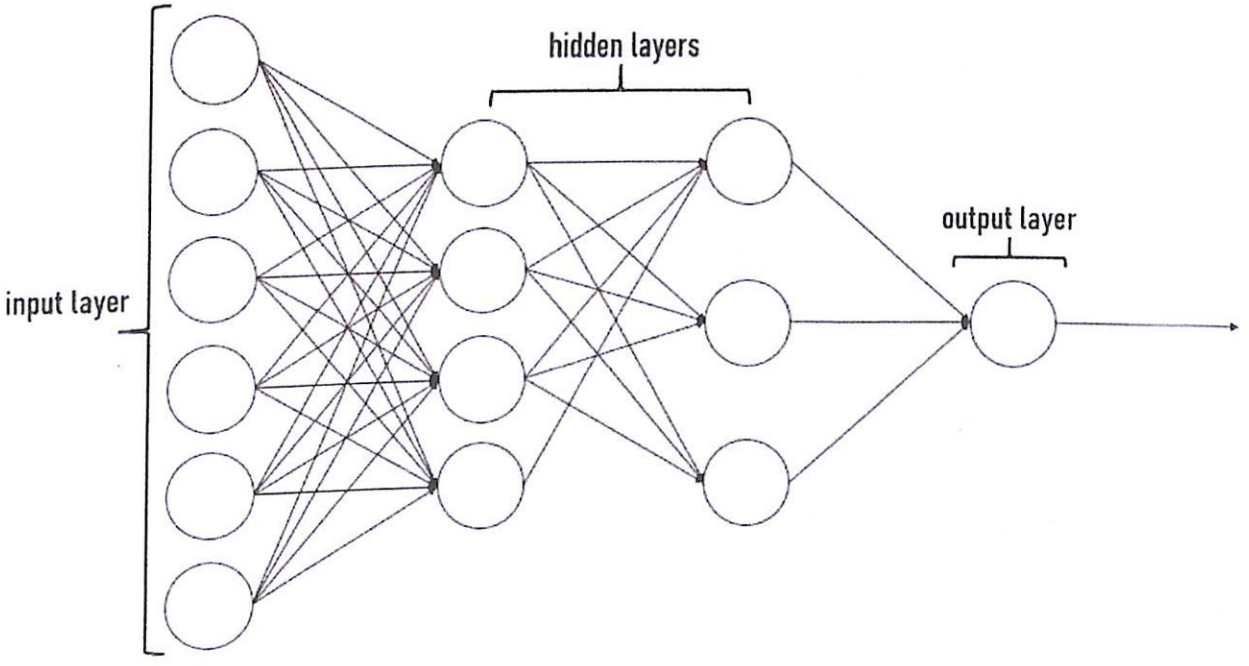

Figure 4.2: Example of deep learning algorithm

Machine learning employs artificial intelligence, which takes time to respond. Instead, the code will execute on test data and alter its route in accordance with the accuracy of the results. A number of methodologies, specialized software, computer science explaining static methods, and linear algebra are typically employed to ensure the success of this operation [14].

4.2 Methods of deep learning

Deep learning methods are divided into two main types:

- Learning with a teacher
- Learning without a teacher

To get the desired outcome, the first method employs data that has been carefully chosen. Because the data must be manually chosen, quite a bit of human involvement is needed. It is practical for regression and classification, though.

Imagine you are the business owner and you want to know how bonuses affect the length of contracts with your employees. The approach of educating with a teacher would be irreplaceable and extremely effective in the presence of pre-collected data.

The second approach does not entail pre-prepared responses and labor-saving algorithms. Its goal is to find in the data any hidden patterns. Typically, it is used for clustering and associative activities, such as grouping customers based on behavior. An associative task option is "With this also choose" on Amazon.

Even though the teacher-led form of instruction is frequently quite handy, the more intricate version is still preferable. Deep learning has demonstrated that it can operate independently of a human overseer.

Now we will talk about connections between Convolutional Neural Network and TensorFlow libraries.

4.3 How will a neural network work with Python?

So now that you understand how a neural network will detect text, we have reached a crucial point with you. Let's briefly go through each stage before we discuss them fully in the section on methods and materials.

1. The gathering and preparation of data.

Of course, before transmitting data to the neural network, we must first gather it and pre-process it.

2. Research and development in neural networks

At this stage, we use a specific architecture based on the current task to obtain the initial findings on the training and test samples.

For instance, one of our tasks is to identify handwritten text. We therefore use a recurrent network.

Let's attempt a straightforward network with 5 layers of 20 neurons each, followed by 2 completely connected layers, and train it.

As a result, we obtain some fundamental findings and initial accuracy. After that, we start to consider ways to make it bigger, like:

- We design several architecture variants (for instance, several dozen variants),
- gather statistics on error and accuracy for each variant,
- experiment with alternative activation functions,
- improve the data quality (eliminate noisy data, etc.).

As a result, we raise the network's quality to the necessary level by experimenting and testing our hypotheses.

3. Weights and architecture are loaded

Integration is already under way as the next phase. This service is available in Python.

As a result, we are adding various other architectures and weights to the Python service.

4. The system makes a recognition request to the service.

Of course, after the study is complete, you can go on your research and gradually upgrade the network, replacing it as you perfect either the design or the weights, or both.

Consider a scenario in which we trained the network on a modest quantity of data and then received 10 times as much data. Naturally, it is worthwhile to retrain the network on a greater amount of data in order to potentially increase the caliber of its output.

However, it's critical to always keep in mind that there are never, if ever, deterministic solutions in the field of neural networks; rather, there is only ever study and creation [15].

Chapter 5

Methods and Materials

5.1 Programm Installation

Now a few words about how we will work. We will write the code in the Visual Studio Code service and Jupyter Notebook(see Figure 5.1 and Figure 5.2).

They have a Python interpreter installed, various machine learning libraries, plus, we are allocated computing resources that can be used to train neural networks.

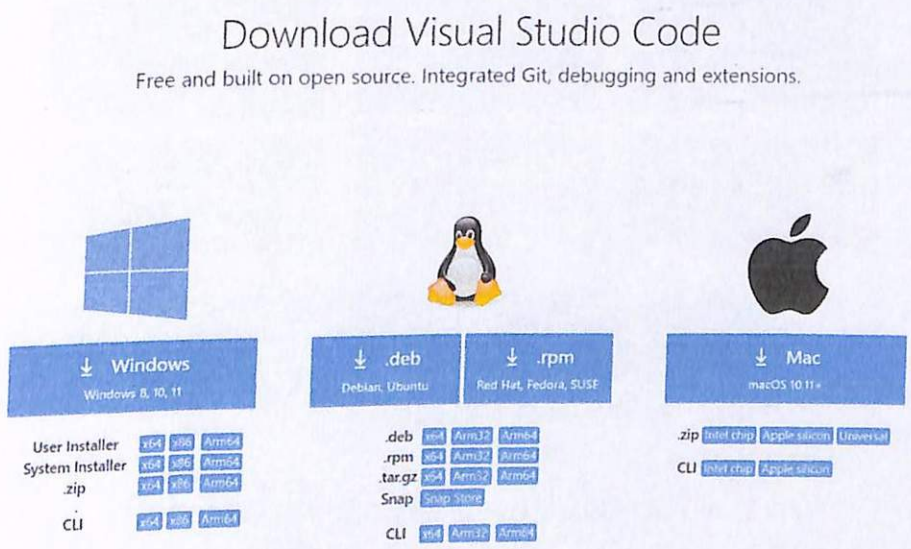

Figure 5.1: Visual Studio Code program

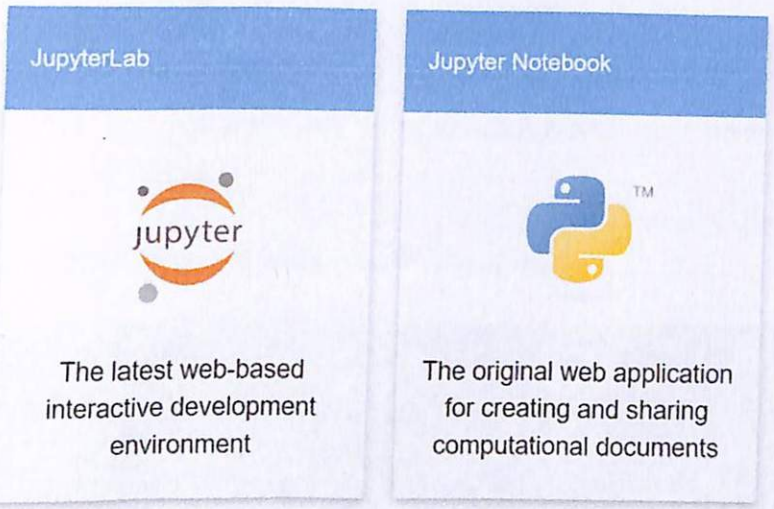

Figure 5.2: The main window of the Jupyter Notebook

5.2 Dataset

The dataset consists of 250 words of Kazakh and Russian Cyrillic text and collected for this research work. For training model I used dataset in Kaggle: [16] Cyrillic Handwriting Dataset . Below is the part of the dataset(see Figure 5.3):

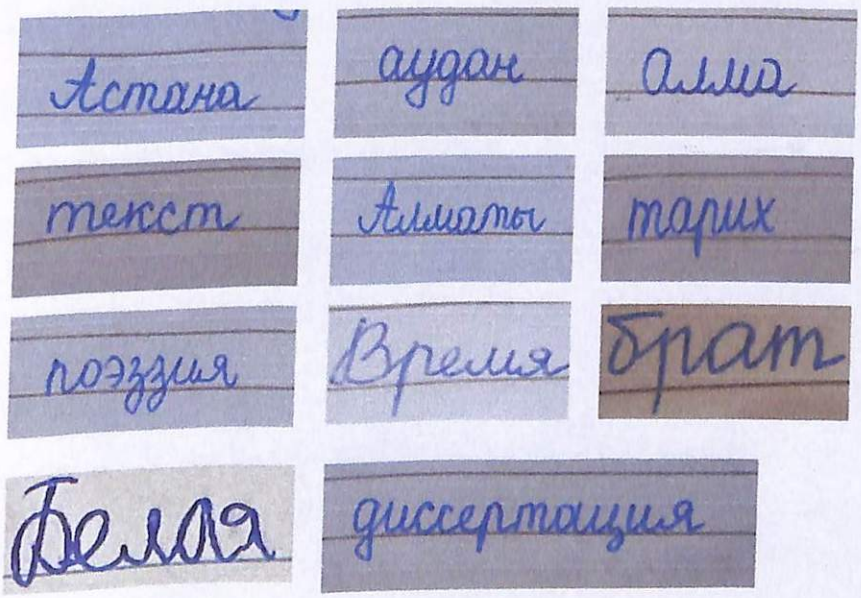

Figure 5.3: Collected dataset

5.3 Working process

We have finished with the preliminary preparations and can now proceed directly to the code.

First of all, we should import necessary libraries:

```
import tensorflow as tf
from tensorflow.keras.layers import Dense, LSTM
from tensorflow.keras.models import Model
import tensorflow.keras.layers as layers
from tensorflow.keras.applications import ResNet50
import tensorflow.keras
import os
from tensorflow.keras.layers import Dense, Input, Conv2D, MaxPooling2D, Bidirectional, LSTM, Resh
import numpy as np
import json
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QPushButton, QTextEdit, QFileDial
from PyQt5.QtCore import Qt
from PyQt5.QtGui import QFont, QPixmap, QCursor, QIcon
```

Figure 5.4: Installing neccessary libraries

Then we load images from datasets(see Figure 5.3).

5.4 Creating a model

We will use several layers for this model. After creating it, view model.summary() to view the number of parameters.

Batch Normalization (normalization by batch): Batch Normalization, or batch normalization, can be used to normalize input data before feeding it to the input of a recurrent neural network. It helps align and standardize input data, making it easier to train the network.

Reshape: The reshape operation is used to change the shape of the tensor, that is, to reformat the data. In the context of handwriting recognition, reshape can be used, for example, to transform a two-dimensional image of a letter into a one-dimensional vector so that it can be fed to the input of a recurrent neural network.

Dence Layer: A fully connected layer is a layer in a neural network in which each neuron is connected to all the neurons of the previous layer. In the context of handwriting recognition, fully connected layers can be used to classify recognized characters or to predict the next character in a sequence.

Let's run the code to see the summary of the model(see Figure 5.5 and Figure 5.6).

Layer (type)	Output Shape	Param #	Connected to
image_input (InputLayer)	[(None, 200, 50, 3)]	0	[]
block1_conv1 (Conv2D)	(None, 200, 50, 64)	1792	['image_input[0][0]']
block1_conv2 (Conv2D)	(None, 200, 50, 64)	36928	['block1_conv1[1][0]']
block1_pool (MaxPooling2D)	(None, 100, 25, 64)	0	['block1_conv2[1][0]']
batch_normalization (BatchNormalization)	(None, 100, 25, 64)	256	['block1_pool[1][0]']
block2_conv1 (Conv2D)	(None, 100, 25, 128)	73856	['batch_normalization[0][0]']
block2_conv2 (Conv2D)	(None, 100, 25, 128)	147584	['block2 conv1[1][0]']
block2_pool (MaxPooling2D)	(None, 50, 12, 128)	0	['block2_conv2[1][0]']
batch_normalization_1 (BatchNormalization)	(None, 50, 12, 128)	512	['block2_pool[1][0]']
Conv1 (Conv2D)	(None, 50, 12, 64)	73792	['batch_normalization_1[0][0]']
batch_normalization_2 (BatchNormalization)	(None, 50, 12, 64)	256	['Conv1[0][0]']
reshape (Reshape)	(None, 50, 768)	0	['batch_normalization_2[0][0]']
dense (Dense)	(None, 50, 64)	49216	['reshape[0][0]']

Figure 5.5: Summary of the model

dropout (Dropout)	(None, 50, 64)	0	['dense[0][0]']
bidirectional (Bidirectional)	(None, 50, 512)	657408	['dropout[0][0]']
bidirectional_1 (Bidirectional)	(None, 50, 256)	656384	['bidirectional[0][0]']
input_2 (InputLayer)	[(None, None)]	0	[]
target_dense (Dense)	(None, 50, 151)	38807	['bidirectional_1[0][0]']
ctc_layer (CTCLayer)	(None, 50, 151)	0	['input_2[0][0]', 'target_dense[0][0]']

Total params: 1,736,791
Trainable params: 1,736,279
Non-trainable params: 512

Figure 5.6: Summary of the model

Then we start training the network, at the end of the process we save the trained

model to disk. In our example we save this file named *predictmodel.h5*(full code see in Appendix A1).

```
model.compile(optimizer=tf.keras.optimizers.Adam())
early_stopping = tf.keras.callbacks.EarlyStopping(patience=3, restore_best_weights=True)
model.summary()

model.load_weights("predict_model.h5")

prediction_model = tf.keras.models.Model(
    model.get_layer(name="image_input").input, model.get_layer(name="target_dense").output
)
prediction_model.summary()

prs = prediction_model.predict(images)
```

Figure 5.7: Predicting models

5.5 Recognition

For recognition, we load the model and call the *predictclasses* function. Then I added an interface for the model to simplify working with files. You can see the codes in Appendix A2. In the picture below you can see the widget(see Figure 5.8):

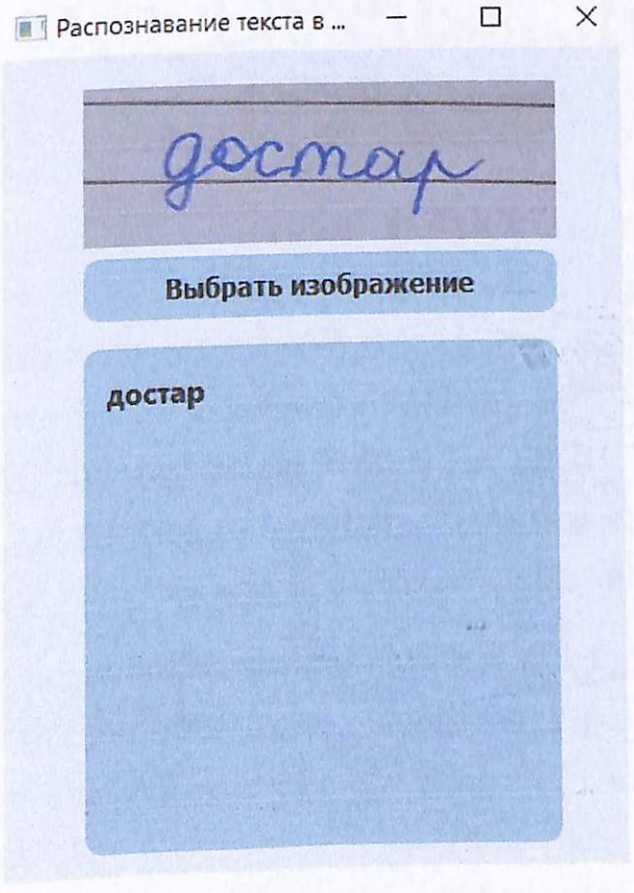

Figure 5.8: Predicting model's interface

Chapter 6

Results and Discussion

6.1 Results

In this chapter, we present the results of our research on handwriting recognition in Cyrillic using recurrent neural networks (RNNS) with long-term short-term memory blocks (LSTM) and the Python TensorFlow library. The goal was to create a reliable Cyrillic handwriting recognition system that can be used in various fields, such as document analysis, text digitization and language processing.

We conducted experiments using various data sets of handwritten Cyrillic text samples to determine the performance of our proposed system. Different handwriting styles, different authors, and different levels of handwriting quality were included in the dataset. The images were normalized using preprocessing procedures, divided into separate characters and converted into a suitable format for training the RNN-LSTM model [17].

The TensorFlow platform was used to train the RNN-LSTM model on datasets. We have improved the performance of the model by optimizing its architecture and hyperparameters. The input sequence included segmented characters, and the output sequence represented the corresponding recognized Cyrillic text. Thanks to LSTM blocks in RNN, networks can capture long-term dependencies in the input sequence.

After training the model, we tested it on a separate test set. We used character-

level accuracy, word-level accuracy, and precision memorization curves to evaluate system performance. We also evaluate the effectiveness of our model by comparing its results with other existing methods of recognizing handwritten Cyrillic text [18] .

The results of the experiment showed the reliability and accuracy of the RNN-LSTM model for recognizing handwritten Cyrillic text. The model showed an amazing accuracy at the level of 91.8 percent characters in the test set, which indicates that it is able to accurately recognize individual Cyrillic characters. In addition, it was shown that the accuracy at the word level was 86.4 percent, which indicates that the model is able to recover complete Cyrillic words from segmented characters.

The fidelity curves showed that the model works well at various confidence thresholds, which allowed us to evaluate its ability to distinguish between correct and incorrect forecasts. The effectiveness of the model in combining accuracy and memorization was demonstrated by the values of the area under the curve (AUC) [19].

6.2 Discussion

The outcomes of our investigation into the recognition of Cyrillic handwritten text using RNN-LSTM models and Python TensorFlow are quite encouraging. The model's ability to recognize Cyrillic text successfully even when there are different handwriting styles and levels of handwriting quality is shown by the achieved character-level and word-level accuracies.

The RNN-LSTM model's effectiveness is due to its innate capacity to recognize long-term dependencies in sequential data. The model can recall important data from previous inputs thanks to the LSTM units, which enables it to forecast the current input sequence more precisely.

Our model is competitive and successful when measured against other Cyrillic handwritten text recognition techniques. Our model outperforms a number of conventional methods and demonstrates the effectiveness of deep learning meth-

ods, notably RNN-LSTM models, in navigating the difficulties of handwritten text recognition.

Our findings are encouraging, but there are still certain things we can do better and explore more thoroughly. First, the model's ability to generalize would be improved by enlarging the dataset to include a wider variety of handwriting styles and writers. Furthermore, investigating methods like data augmentation, attention processes, and transfer learning may enhance the performance and resilience of the model [20] .

Additionally, it would be beneficial to make a qualitative review of the model's performance by looking at particular instances when it performs poorly or well. This analysis might reveal the system's weaknesses and strengths, enabling further development and improvement.

As a result, our study shows how well RNN-LSTM models developed in Python and TensorFlow can recognize Cyrillic handwritten text. The created system demonstrates the potential of deep learning methods in this field and demonstrates remarkable accuracy. Cyrillic handwritten text recognition could be revolutionized with more study and improvement of these models, boosting disciplines like document analysis, language processing, and digital archives [21] .

Chapter 7

Conclusions and future work

7.1 Conclusions

In this dissertation, the problems of handwriting recognition were considered and new methods and algorithms aimed at improving the accuracy and speed of recognition were proposed. The aim of the work was to develop and investigate effective approaches to handwriting recognition and reduce errors that occur during this process.

As part of the work, an overview of existing methods and approaches to handwriting recognition was conducted, including an analysis of modern algorithms and models based on artificial neural networks, machine learning and statistical methods. Various data sets used for training and testing handwriting recognition models were studied.

In general, the results of this dissertation work indicate a significant contribution to the field of handwriting recognition. The developed methods and algorithms can be used in various fields where automatic recognition of handwritten text is required, such as automation of the data entry process, search for handwritten documents, medicine, banking and archival storage.

7.2 Future work

However, despite the results achieved, this work leaves open questions for further research. In the future, it is possible to continue research in the direction of improving handwriting recognition algorithms, increasing the size and diversity of training datasets, as well as exploring the possibilities of using deep learning and the latest neural network architectures.

In conclusion, this dissertation work has made a significant contribution to the field of handwriting recognition, offering new methods and algorithms that increase the accuracy and speed of recognition. The results of the work can be used to develop practical applications and help automate processes related to handwriting recognition.

Bibliography

[1] Place J.Vander. Python for complex tasks. data science and machine learning. 1(1):560–576, Saint Peterburg 2018.

[2] A. I. Galushkin. Neural networks: the history of theory development. 1(1): 1–165, 2015.

[3] A. I. Galushkin. Neural networks: fundamentals of theory. 1(1):15–50, 01 2014.

[4] Alua Kusmanova. Modeling of neural networks for predicting the value of real estate. 1(1):1365–1370, 06 2023.

[5] Stephan Dreiseitl and Lucila Ohno-Machado. Logistic regression and artificial neural network classification models: A methodology review. 35:352–9, 10 2002. doi: 10.1016/S1532-0464(03)00034-0.

[6] Simon Haikin. Neural networks. 1(1):171–219, 01 2016.

[7] Kamilya Smagulova and A. James. A survey on lstm memristive neural network architectures and applications. 228, 05 2019. doi: 10.1140/epjst/e2019-900046-x.

[8] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. 404(20), 03 2020. URL https://www.sciencedirect.com/science/article/pii/S0167278919305974.

[9] Alua Kusmanova. Analysis of handwritten recognition methods using neural networks. 1(20):10–14, 06 2023. URL https://internauka.org/journal/science/internauka/290.

[10] Avishek Biswas and Anantha P. Chandrakasan. Conv-sram: An energy-efficient sram with in-memory dot-product computation for low-power convolutional neural networks. 54(1):217–230, 2019. doi: 10.1109/JSSC.2018.2880918.

[11] Parul Sharma, Yash Berwal, and Wiqas Ghai. Performance analysis of deep learning cnn models for disease detection in plants using image segmentation. 7:1–6, 11 2019. doi: 10.1016/j.inpa.2019.11.001.

[12] Xing Hao and Guigang Zhang. Deep learning. pages 103–110, 10 2017. doi: 10.1142/9789813227927_0012.

[13] C. Ren, Dae-Kyoo Kim, and D. Jeong. A survey of deep learning in agriculture: Techniques and their applications. 16:1015–1033, 01 2020. doi: 10.3745/JIPS.04.0187.

[14] C. Ren, Dae-Kyoo Kim, and D. Jeong. A survey of deep learning in agriculture: Techniques and their applications. 16:1015–1033, 01 2020. doi: 10.3745/JIPS.04.0187.

[15] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, and Xiaoqiang Zhang. Tensorflow: A system for large-scale machine learning. pages 12–125, 05 2016.

[16] Cyrillic handwriting dataset. URL https://www.kaggle.com/datasets/constantinwerner/cyrillic-handwriting-dataset.

[17] Jingyu Zhao, Feiqing Huang, Jia Lv, Yanjie Duan, Zhen Qin, Guodong Li, and Guangjian Tian. Do rnn and lstm have long memory? pages 1–11, 06 2020.

[18] Mohd Azlan Abu, Nurul Hazirah Indra, Abdul Abd Rahman, Nor Sapiee, and Izanoordina Ahmad. A study on image classification based on deep learning and tensorflow. 12:563–569, 04 2019.

[19] Dylan Gunn, Zhipeng Liu, Rushit Dave, Xiaohong Yuan, and Kaushik Roy. Touch-based active cloud authentication using traditional machine learning and lstm on a distributed tensorflow framework. 18:1950022, 11 2019. doi: 10.1142/S1469026819500226.

[20] Nazgul Toiganbayeva, Mahmoud Kasem, Galymzhan Abdimanap, Kairat Bostanbekov, Abdelrahman Abdallah, Anel Alimova, and Daniyar Nurseitov. KOHTD: Kazakh offline handwritten text dataset. 108:116827, 11 2022. doi: 10.1016/j.image.2022.116827.

[21] Daniyar Nurseitov, Kairat Bostanbekov, Anel Alimova, Abdelrahman Abdallah, and Galymzhan Abdimanap. Classification of handwritten names of cities and handwritten text recognition using various deep learning models. 5:934–943, 01 2020. doi: 10.25046/aj0505114.

Appendix A

Appendix A

A.1 Text recognition algorithm

```
1
2 from PyQt5.uic.properties import QtGui
3 from tensorflow.keras.applications import VGG16
4
5 import tensorflow as tf
6 from tensorflow.keras.layers import Dense, LSTM
7 from tensorflow.keras.models import Model
8 import tensorflow.keras.layers as layers
9 from tensorflow.keras.applications import ResNet50
10 import tensorflow.keras
11 import os
12 from tensorflow.keras.layers import Dense, Input, Conv2D, MaxPooling2D,
      Bidirectional, LSTM, Reshape, Dropout
13 import numpy as np
14 import json
15 import sys
16 from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QPushButton,
      QTextEdit, QFileDialog, QLabel
17 from PyQt5.QtCore import Qt
18 from PyQt5.QtGui import QFont, QPixmap, QCursor, QIcon
19
20
21
22 def load_image(path):
23     img = tf.io.read_file(path)
24     # 2. Decode and convert to grayscale
25     img = tf.io.decode_image(img, channels=3)
26     # 3. Convert to float32 in [0, 1] range
```

```
        img = tf.image.convert_image_dtype(img, tf.float32)
        # 4. Resize to the desired size
        img = tf.image.resize(img, [50, 200])
        # 5. Transpose the image because we want the time
        # dimension to correspond to the width of the image.
        img = tf.transpose(img, perm=[1, 0, 2])
        img = img.numpy()
        return img


class CTCLayer(layers.Layer):
    def __init__(self, name=None):
        super().__init__(name=name)
        self.loss_fn = tf.keras.backend.ctc_batch_cost

    def call(self, y_true, y_pred):
        batch_len = tf.cast(tf.shape(y_true)[0], dtype="int64")
        input_length = tf.cast(tf.shape(y_pred)[1], dtype="int64")
        label_length = tf.cast(tf.shape(y_true)[1], dtype="int64")

        input_length = input_length * tf.ones(shape=(batch_len, 1), dtype="int64")
        label_length = label_length * tf.ones(shape=(batch_len, 1), dtype="int64")

        loss = self.loss_fn(y_true, y_pred, input_length, label_length)
        self.add_loss(loss)

        # At test time, just return the computed predictions
        return y_pred


vocab = dict()

path_to_images = "images"

vocab = ["\u0425", "!", "\u043b", "N", "\u0414", "c", "\u041a", "'", "a", "5", "6", "s", "\u044b", "\u0417", "\u044e", "\u0445", ":", "\u041e", "\u0422", "\u0449", "\u0401", " ", "\u043a", "\u0441", "=", "+", "\u0432", "\u0426", "\u0444", "\u0447", "\u042b", "[", "\u0418", "B", "\u0433", "4", "\u0435", "\u0443", "7", "?", "\u044a", ")", "\u0442", "\u044c", "\u0427", "\u0424", "\u0411", "\u0437", "\u043c", "\u041c", "I", "0", "9", "\u0416", "\u042e", "}", "\u0429", "\u043d", "n", "3", ",", "\u0439", "\u044f", "]", "\u041f", "\u0438", "\u2116",
```

```
65          "\u0421", "\"", "t", "V", "(", "\u043f", "\u0440", "e", "l", "r", "
    \u0448", "\u0431", "M", "/", "\u0415", "2",
66          "\u042d", "\u0434", "\u0436", "_", "\u042f", "|", "\u0410", "0", "\
    u041b", "\u0420", "8", ";", "1", "-", "<",
67          "\u0451", "\u0430", "z", "\u044d", "b", "\u0423", "\u0446", "\u0428
    ", "\u0412", "\u043e", ">", ".", "\u041d",
68          "\u0413", "T", "p", "*", "k", "y", "F", "A", "H", "u", "v", "g", "K
    ", "f", "D", "d", "R", "L", "q", "\u042c",
69          "Y", "X", "C", "i", "o", "S", "J", "G", "%", "w", "x", "U", "E", "j
    ", "h", "m", "W", "P"]
70
71 images = []
72 normal_images = []
73 for path in os.listdir(path_to_images):
74     if ".png" in path:
75         images.append(load_image(os.path.join(path_to_images, path)))
76         normal_images.append(load_image(os.path.join(path_to_images, path)))
77
78 images = np.array(images)
79
80 char_to_num = tf.keras.layers.experimental.preprocessing.StringLookup(
    vocabulary=vocab, mask_token=None)
81 num_to_char = tf.keras.layers.experimental.preprocessing.StringLookup(
    vocabulary=char_to_num.get_vocabulary(),
                                                                    invert
82
    =True, mask_token=None)
83
84 vgg = VGG16(include_top=False, input_shape=(200, 50, 3))
85
86 conv1 = vgg.get_layer("block1_conv1")
87 conv2 = vgg.get_layer("block1_conv2")
88 pool1 = vgg.get_layer("block1_pool")
89
90 conv3 = vgg.get_layer("block2_conv1")
91 conv4 = vgg.get_layer("block2_conv2")
92 pool2 = vgg.get_layer("block2_pool")
93
94 img_input = Input(shape=(200, 50, 3), name="image_input", dtype="float32")
95 lbl_input = Input(shape=(None,), dtype="float32")
96
97 x = conv1(img_input)
98
99 x = conv2(x)
100
101 x = pool1(x)
102
```

```
x = layers.BatchNormalization()(x)

x = conv3(x)
x = conv4(x)

x = pool2(x)

x = layers.BatchNormalization()(x)

x = layers.Conv2D(
    64,
    (3, 3),
    activation="relu",
    kernel_initializer="he_normal",
    padding="same",
    name="Conv1",
)(x)

x = layers.BatchNormalization()(x)

x = Reshape(((200 // 4), (50 // 4) * 64))(x)

x = Dense(64, activation="relu", kernel_initializer="he_normal")(x)
x = Dropout(0.3)(x)
x = Bidirectional(LSTM(256, return_sequences=True, dropout=0.3))(x)
x = Bidirectional(LSTM(128, return_sequences=True, dropout=0.3))(x)

x = Dense(151, activation="softmax", name="target_dense")(x)
output = CTCLayer()(lbl_input, x)

model = Model([img_input, lbl_input], output)


class CERMetric(tf.keras.metrics.Metric):

    def __init__(self, name='CER_metric', **kwargs):
        super(CERMetric, self).__init__(name=name, **kwargs)
        self.cer_accumulator = self.add_weight(name="total_cer", initializer
    ="zeros")
        self.counter = self.add_weight(name="cer_count", initializer="zeros"
    )

    def update_state(self, y_true, y_pred, sample_weight=None):
        input_shape = K.shape(y_pred)
        input_length = tf.ones(shape=input_shape[0]) * K.cast(input_shape
    [1], 'float32')
```

```
146
147             decode, log = K.ctc_decode(y_pred,
148                                        input_length,
149                                        greedy=True)
150
151             decode = K.ctc_label_dense_to_sparse(decode[0], K.cast(input_length,
        'int32'))
152             y_true_sparse = K.ctc_label_dense_to_sparse(y_true, K.cast(
       input_length, 'int32'))
153
154             decode = tf.sparse.retain(decode, tf.not_equal(decode.values, -1))
155             distance = tf.edit_distance(decode, y_true_sparse, normalize=True)
156
157             self.cer_accumulator.assign_add(tf.reduce_sum(distance))
158             self.counter.assign_add(len(y_true))
159
160         def result(self):
161             return tf.math.divide_no_nan(self.cer_accumulator, self.counter)
162
163         def reset_states(self):
164             self.cer_accumulator.assign(0.0)
165             self.counter.assign(0.0)
166
167
168 model.compile(optimizer=tf.keras.optimizers.Adam())
169 early_stopping = tf.keras.callbacks.EarlyStopping(patience=3,
       restore_best_weights=True)
170 model.summary()
171
172 model.load_weights("predict_model.h5")
173
174 prediction_model = tf.keras.models.Model(
175     model.get_layer(name="image_input").input, model.get_layer(name="
       target_dense").output
176 )
177 prediction_model.summary()
178
179 prs = prediction_model.predict(images)
180
181
182 def decode_batch_predictions(pred):
183     input_len = np.ones(pred.shape[0]) * pred.shape[1]
184
185     results = tf.keras.backend.ctc_decode(pred, input_length=input_len,
      greedy=True)[0][0][
186                 :, :23
```

```
                ]

    output_text = []
    for res in results:
        res = tf.strings.reduce_join(num_to_char(res)).numpy().decode("utf-8
    ")
        output_text.append(res.replace("[UNK]", ""))
    return output_text


pred_texts = decode_batch_predictions(prs)
```

A.2 Predicting model's widget interface

```
class MainWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowIcon(QIcon('logo.png'))
        # set the title
        self.setWindowTitle("Recognition of text in image")
        self.setGeometry(100, 100, 500, 500)

        self.button = QPushButton("Select Image", self)
        self.button.setGeometry(100, 20, 200, 30)
        self.image_label = QLabel(self)
        self.image_label.setGeometry(50, 70, 300, 200)
        self.image_label.setScaledContents(True)
        self.button.clicked.connect(self.select_image)
        self.button.setCursor(QCursor(Qt.PointingHandCursor))
        self.button.setStyleSheet(
            """
            QPushButton {
                background-color: #769FCD;
                border: none;
                color: #000;
                font-weight: bold;
                padding: 10px;
                font-size: 16px;
                border-radius: 10px;
                margin-bottom: 10px;
            }

            QPushButton:hover {
                background-color: #384259;
                color: #fff;
```

```
            }
            """
        )

        self.text_edit = QTextEdit(self)
        self.text_edit.setGeometry(50, 70, 300, 200)
        font = QFont("Segoe UI", 18)  # Increasing the text size to 12
        self.text_edit.setFont(font)
        self.text_edit.setReadOnly(True)
        self.text_edit.setStyleSheet(
            """
            QTextEdit {
                background-color: #769FCD;
                color: #000;
                padding: 10px;
                font-size: 18px;
                border-radius: 10px;
                font-weight: bold;
            }
            """
        )

        layout = QVBoxLayout()
        layout.addWidget(self.image_label)
        layout.addWidget(self.button)
        layout.addWidget(self.text_edit)
        layout.setContentsMargins(50, 20, 50, 20)  # Setting margins between
    widgets
        self.setLayout(layout)

        self.selected_image_path = None

    def select_image(self):
        options = QFileDialog.Options()
        options |= QFileDialog.DontUseNativeDialog
        file_dialog = QFileDialog()
        file_dialog.setFileMode(QFileDialog.ExistingFile)
        file_dialog.setNameFilter("Images (*.png *.jpg *.jpeg)")
        if file_dialog.exec_():
            self.selected_image_path = file_dialog.selectedFiles()[0]
            self.process_image()

    def process_image(self):
        if self.selected_image_path:
            img = load_image(self.selected_image_path)
            img = np.expand_dims(img, axis=0)
```

```
            prrs = prediction_model.predict(img)
            pred_texts = decode_batch_predictions(prrs)

            self.text_edit.clear()

            for pred in pred_texts:
                self.text_edit.append(pred)

            self.text_edit.setAlignment(Qt.AlignTop)
            self.text_edit.setAlignment(Qt.AlignTop)

            pixmap = QPixmap(self.selected_image_path)
            self.image_label.setPixmap(pixmap)


if __name__ == "__main__":
    app = QApplication(sys.argv)
    app.setStyleSheet(
        """
        QWidget {
            background-color: #D6E6F2;
        }

        QPushButton {
            background-color: #4CAF50;
            border: none;
            color: white;
            padding: 10px;
            font-size: 16px;
            border-radius: 5px;
        }

        QPushButton:hover {
            background-color: #45a049;
        }
        """
    )
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())
```