FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Text recognition in historical archival material from the 17th-19th centuries
Student:	Herman Tiumentsev
Supervisor:	Ing. Jiří Smítka
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2022/2023

Instructions

Tématem práce je vytvoření prostředí pro testování algoritmů na rozpoznávání historických dokumentů.
Proveďte rešerši používaných algoritmů pro rozpoznávání textu.
Navrhněte a implementujte testovací prostředí, které umožní porovnat kvality různých algoritmů.
Sestavte testovací a trénovací data a ověřte na nich vybrané algoritmy.
K tvorbě datasetu využijte digitální knihovnu Kramerius, kterou provozuje Národní knihovna ČR.

Electronically approved by Ing. Karel Klouda, Ph.D. on 8 April 2021 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Text recognition in historical archival material from the 17th-19th centuries

Herman Tiumentsev

Department of Applied Mathematics
Supervisor: Ing. Jiří Smítka

June 27, 2021

Acknowledgements

I would like to express my appreciation for the guidance from the supervisor of this thesis Ing. Jiří Smítka. I also thank all my friends and family for supporting me in my time of need.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on June 27, 2021 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology
© 2021 Herman Tiumentsev. All rights reserved.
This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis

Tiumentsev, Herman. *Text recognition in historical archival material from the 17th-19th centuries*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstrakt

Digitalizace historických dokumentů je důležitým úkolem, který může badatelům pomoci pohodlnější přistupovat k historickým datům. Cílem této práce je taková digitalizace týkající se dat z digitální knihovny Kramerius provozované Národní knihovnou České republiky. Obsahuje celou pipeline, která se skládá z vytvoření datové sady, různých technik předzpracování dat, dalšího trénování modelů digitálního rozpoznávání (například Tesseract LSTM, Tesseract Legacy nebo GOCR) a vyhodnocení výsledků. Navíc, tato práce navrhuje novou metriku, kterou lze použít k vyhodnocení účinnosti modelu ve srovnání s referenčním modelem.

Klíčová slova rozpoznávání textu, OCR, historické dokumenty, porovnání výkonu, testování OCR algoritmů, zpracování obrazu

Abstract

Historical document digitization is an important task that might help researchers to access historical data more conveniently. This thesis aims at such digitalization related to the data from the Kramerius digital library operated by the National Library of the Czech Republic. It contains the whole pipeline that consists of dataset creation, various data preparation techniques, further digital recognition model training (such as Tesseract LSTM, Tesseract Legacy, and GOCR), and evaluation of the results. Moreover, it proposes a new metric that can be used to evaluate the efficiency of a model comparing to the reference one.

Keywords Text recognition, OCR, Historical documents, Performance comparison, Testing OCR algorithms, Image processing

Contents

List of Tables **xv**

Introduction **1**

1 Image processing **3**
1.1 Grayscaling . . . 4
1.2 Noise reduction . . . 4
1.2.1 Linear filters . . . 4
1.2.2 Median filter . . . 5
1.2.3 Gaussian filter . . . 5
1.3 Area segmentation . . . 6
1.3.1 Binary image . . . 6
1.3.2 Thresholding . . . 6
1.3.3 Otsu's method . . . 6
1.3.4 U-Net . . . 8
1.4 Mathematical morphology . . . 9
1.4.1 Translation . . . 9
1.4.2 Dilation . . . 10
1.4.3 Erosion . . . 10
1.4.4 Closing . . . 10
1.4.5 Opening . . . 11
1.5 Feature extracting . . . 13
1.5.1 Edge detection . . . 13
1.5.2 Gradient . . . 14
1.5.3 Prewitt operator . . . 14
1.5.4 Sobel operator . . . 14
1.5.5 Non-maximums suppression . . . 15
1.5.6 Hysteresis thresholding . . . 16
1.5.7 Canny edge detector . . . 16

2 Soft computing methods **19**
2.1 Artificial neuron . . . 19
2.2 Activation function . . . 19
2.3 Loss function . . . 20
2.4 Cross-entropy . . . 20
2.5 Intersection over Union metric . . . 20
2.6 Backpropagation . . . 20
2.7 Convolution . . . 21
2.8 Convolutional layer . . . 21
2.9 Pooling layer . . . 23
2.10 Softmax nonlinearity . . . 24
2.11 Recurrent neural network . . . 24
2.12 Long short-term memory . . . 25
2.13 Beam search . . . 27
2.14 Transfer Learning . . . 28

3 OCR software **29**
3.1 Commercial . . . 29
3.1.1 ABBYY FineReader . . . 29
3.1.2 OmniPage . . . 29
3.1.3 Adobe Acrobat Pro DC . . . 31
3.2 Non-commercial . . . 32
3.2.1 Tesseract OCR . . . 32
3.2.1.1 Architecture . . . 33
3.2.1.2 Line finding . . . 33
3.2.1.3 Baseline fitting . . . 33
3.2.1.4 Character chopping . . . 34
3.2.1.5 Word recognition . . . 34
3.2.1.6 Character classification . . . 35
3.2.1.7 LSTM Integration . . . 36
3.2.2 GOCR . . . 37

4 Comparison methods **39**
4.1 Comparison metrics . . . 39
4.2 Algorithm performance presentation . . . 40
4.3 Text similarity . . . 41
4.4 Character Error Rate . . . 42

5 Data extraction **45**
5.1 U-net processing . . . 45
5.1.1 Architecture . . . 45
5.1.2 Training . . . 46
5.1.3 Prediction . . . 46
5.2 Segmentation methods comparison . . . 46

5.3 Forming dataset 49
5.3.1 Splitting page text into lines 49
5.3.2 Book format 50

6 Results **53**
6.1 Training state-of-the art engine 53
6.2 Testing algorithms 54
6.2.1 Tesseract LSTM 55
6.2.2 Tesseract Legacy 55
6.2.3 GOCR 56

Conclusion **59**

Bibliography **61**

A Acronyms **65**

B Contents of enclosed CD **67**

C Appendix **69**

List of Figures

1.1 Image pre-processing pipeline. 3
1.2 Visual representation of the weighted median filter. 5
1.3 Otsu's Thresholding. 7
1.4 U-net architecture. 8
1.5 Translation illustration. 9
1.6 Dilation explanation. 11
1.7 Erosion explanation with a cross-shaped structuring element. . . . 12
1.8 Closing explanation. 13
1.9 Opening explanation with a cross-shaped structuring element. . . . 13
1.10 Operators work illustration. 15
1.11 Illustration of finding of edge direction. 15
1.12 Hysteresis thresholding. 16

2.1 Example of convolution of two 5×5 and 3×3 matrices [1]. 22
2.2 A convolution with a three-dimensional kernel [2]. 22
2.3 Example of a pooling operation with a maximum function [2]. . . . 23
2.4 RNN and its expanded representation. 24
2.5 Diagram of the recurrent network layer. 25
2.6 Diagram of the Long-Term Memory layers. 25
2.7 Forget gate layer. 26
2.8 Input layer gate and tanh laye. 26
2.9 Change of the main "highway". 27
2.10 Change of the secondary "highway". 27

3.1 ABBYY FineReader program. 30
3.2 OmniPage program. 30
3.3 Adobe Acrobat Pro DC program. 31
3.4 Google trends comparison for different open source OCR tools. . . 32
3.5 Tesseract OCR Architecture [3]. 33
3.6 An example of a curved fitted baseline [4]. 34

3.7 A fixed-pitch chopped word [4]. 34
3.8 Some difficult word spacing of non-fxed pitch text [4]. 34
3.9 Candidate chop points and chop [4]. 35
3.10 Associating broken characters [4]. 35
3.11 (a) Original letter 'h', (b) broken 'h', (c) features matched to prototypes [4]. 36
3.12 Tesseract using LSTM [5]. 36
3.13 GOCR processing problems [6]. 37

5.1 Training data for U-net. 46
5.2 U-net training overview with binary crossentropy loss function. . . 47
5.3 U-net training overview with Intersection over Union metric. . . . 47
5.4 Examples of masks predicted by U-net. 48
5.5 Segmentation masks by different algorithms. 48
5.6 Bounding boxes example. 49
5.7 Deskewing image. 50
5.8 Splitting page into lines. 51
5.9 Letters for dataset. 51
5.10 Book as JSON example. 52

6.1 Tesseract training results. 53
6.2 Tesseract metrics as JSON example. 54
6.3 Algorithm evaluation as JSON example. 55
6.4 Tesseract processing result. Green is Tesseract LSTM result, yellow is Tesseract Legacy, uncolored characters both algorithms have the same. 56
6.5 GOCR Processing result. 57

List of Tables

1.1 Otsu's calculating table. 7

4.1 Levenstein transformations table 41
4.2 An example of how the Levenshtein algorithm works. 42

5.1 Evaluation of different segmentation methods. Error based on Levenstein similarity 49

6.1 Tesseract evaluating variables. 55
6.2 Tesseract Legacy testing score 56
6.3 GOCR testing score 57

C.1 U-net model architecture, Total params: 31 030 593. 70

Introduction

Historical document digitization is an important task for the preservation of our cultural heritage. For centuries, an enormous number of important articles, documents, and novels have been collected in the form of paper publications and scanned image files. However, the data accumulated in this format is difficult to analyze and process, and there is no way to search for words and phrases in the text. So the text digitization has become essential for knowledge extracting and informational retrieval from such type of data.

Decades ago, scientists may go through hours in a library looking through a large number of pages to locate a valuable paragraph. In the age of modern technology and the World Wide Web, there are opportunities to classify and collect knowledge from different fields in one place, which makes working with various text sources of information much faster. Things get much simpler when we can work with digitized text.

It is especially difficult to digitize manuscripts. If the processing document is typewritten or handwritten, the digitization process gets more complicated due to a wide range of fonts and a huge amount of noise while writing process.

One of the ways to translate documents into text from is rewriting all the content manually by keyboard typing, but this process will take too much effort and time, so this solution will not be effective.

Modern optical character recognition (OCR) frameworks cope with this kind of task. These algorithms can naturally recognize text and convert it to a readable and editable digital format many times faster than a human.

This thesis exploring a set of methods that allows performing an optical character recognition on scanned documents. There are many different OCR algorithms, so it is important to describe, classify some of them and measure their efficiency. It will reveal the advantages and disadvantages of different algorithms, allowing us to draw some conclusions about the feasibility of their use.

Such innovation underpins the formation of digital library materials and makes accessible huge amount of text over the Internet.

Goals of this thesis are to test different text recognition algorithms and determine a measure of the applicability of various techniques in a historical archive environment using data from the Kramerius digital library operated by the National Library of the Czech Republic.

Chapter 1

Image processing

With digitized texts, people can without much of a stretch quest search for anything they want. OCR innovations were first evolved to peruse printed text, so early present day handwriting presents new difficulties. But before starting to explore OCR algorithms it is necessary to know basic approaches and definitions.

Even the most sophisticated analysis is of no use if it is based on bad data. The phrase "Garbage in, garbage out" [7] uttered by Charles Babbage who lived in the 19th century also applies to data mining and machine learning projects, so data pre-processing is an important step in the data mining processes.

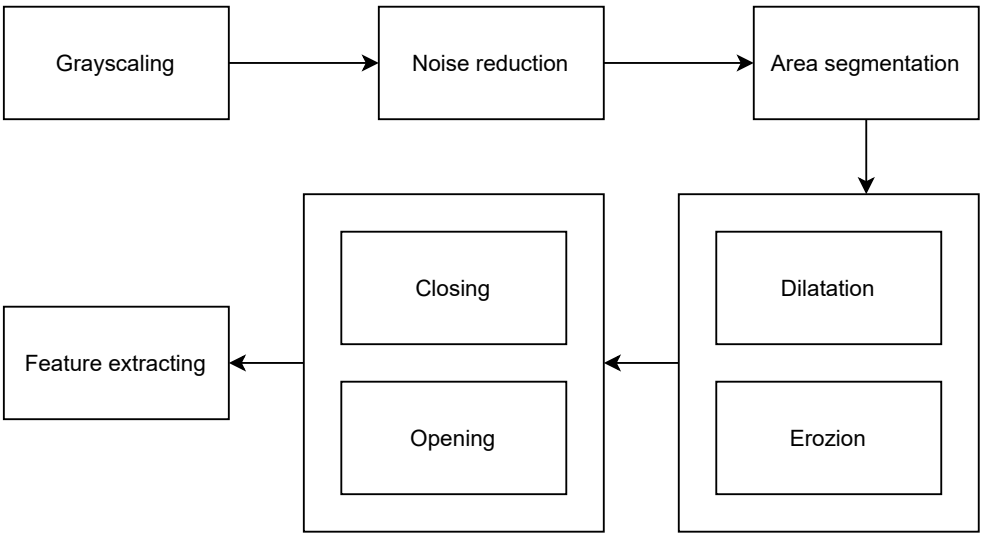


Figure 1.1: Image pre-processing pipeline.

As a field of signal processing, digital image processing has many advantages over analog processing. It allows a much wider range of algorithms to be applied to input data and avoids problems such as added noise and distortion during

processing. With image processing, it is possible to get an enhanced image or extract useful information from it. So image processing can significantly help with recognition of text in a picture. The image 1.1 summarizes the basic principles. The following terms refer to the book [8].

1.1 Grayscaling

Grayscaling is a process of converting colored image into shades of gray, representing the degree of brightness of the white color. The grayscale image consists of only one channel, where each pixel is represented by 8 bits and takes values from 0 (black) to 255 (white color). To make the grayscale image more contrasting, the weighting method is used. Its principle is to add all the values from the three color channels multiplied by the coefficients while converting an image from the RGB model. Because of the peculiarities of human vision, green is multiplied by the biggest coefficient for a high-quality conversion. This is due to the fact that in ancient times our ancestors lived their entire lives outdoors and saw a lot of greenery and plants around, so the human eye sees more green shades. Referring to framework [9] that I used in my experiments, the translation is done as follows:

$$G(x, y) = 0,299 * R + 0,587 * G + 0,114 * B, \tag{1.1}$$

where x and y are Cartesian coordinates describing the image plane.

1.2 Noise reduction

Every image in the form of a natural scene is faced with the presence of some degree of unwanted noise. This can cause problems in further processing. Therefore, it is necessary to remove or at least reduce the noise first. When working with a scanned document containing text of unknown size, it is important to consider techniques that preserve some level of edge detail while reducing the noise. Median and Gaussian filtering cope with such tasks.

1.2.1 Linear filters

The most common type of filters are linear filters, in which the value of the output pixel is determined as a weighted sum of the values of the input pixels.

$$I'(x, y) = \sum_i \sum_j I(x + i, y + j) \cdot H(i, j), \tag{1.2}$$

where I represents original image, h is a kernel, which represents coefficients of the filter. In general this formula can be written as follows:

$$I'(x, y) = I * H, \tag{1.3}$$

where $*$ is a convolution operation [10].

1.2.2 Median filter

Median filter replaces the value of a given pixel by the median of the values of neighboring pixels located in a square vicinity around that pixel. Pixel values inside the filter window are sorted in ascending (descending) order and the value that is in the middle of the ordered list goes to the filter output.

$$I'(x,y) = Me\{I(x+i, y+j)|(i,j) \in R\}. \tag{1.4}$$

A variation of the median filter is the weighted median filter. Such a filter uses a weighting function but it is interpreted differently than in linear filters. The weighting coefficients show how many times pixels of the image, in the vicinity of each individual pixel, should be taken into account in the calculations.

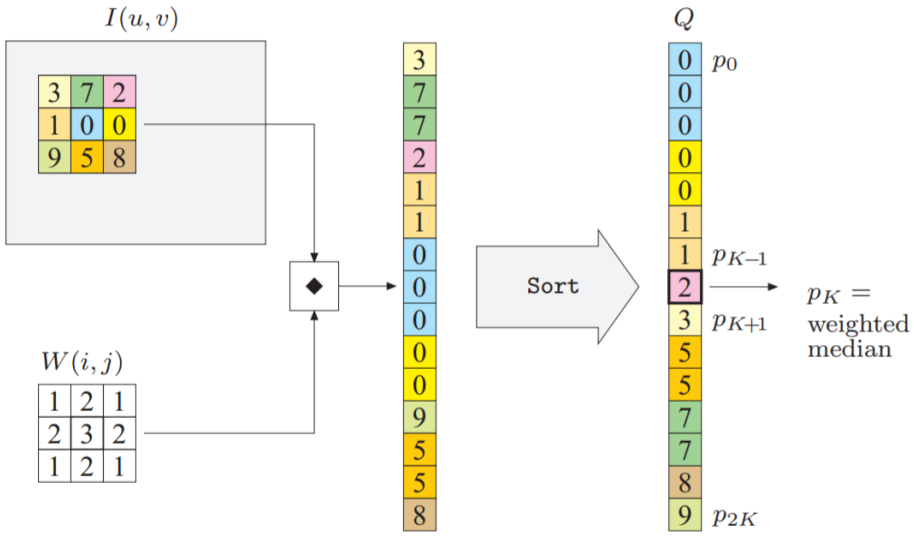


Figure 1.2: Visual representation of the weighted median filter.

1.2.3 Gaussian filter

A Gaussian filter is a linear filter that has the following kernel:

$$H_{gauss}(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}}, \tag{1.5}$$

where σ^2 variance of a random variable. A Gaussian filter has a nonzero kernel of infinite size. However, the filter kernel rapidly decreases to zero with distancing from the point $(0,0)$, and therefore in practice one can be limited to convolution with a window of small size around $(0,0)$ (by taking the radius

of the window as 3σ for example). So the image transformation formula is as follows:

$$I'(x,y) = \frac{1}{2\pi\sigma^2} \sum_i \sum_j e^{-\frac{i^2+j^2}{2\sigma^2}} \cdot I(x+i, y+j). \tag{1.6}$$

1.3 Area segmentation

1.3.1 Binary image

A binary image consists of a single channel where each pixel takes on the value 0 or the value 1. There are lot of ways to convert an image into binary. Images represented by color models or shades of gray can be converted to a binary image by, for example, thresholding. The 0 value is conventionally called the background, and 1 the foreground.

1.3.2 Thresholding

A filter separates objects from the surrounding background when the brightness of the pixels of the objects and the background are concentrated near the two predominant values. Usually the filter is used to obtain a binary image from a grayscale image or to remove noise.

Selection of objects is performed by determining the value of the threshold that separates areas of brightness distribution. If the brightness value of a particular pixel is greater than the threshold value, then that point belongs to the object, if less to the background.

There is also a way of double thresholding which is a variant of simple thresholding, where we use two thresholds t_1 and t_2. If the pixel value is between the threshold values t_1 and t_2, $(t_1 < t_2)$ then in the resulting binary image, the pixel has a value corresponding to white, otherwise it has a value corresponding to the black colour.

1.3.3 Otsu's method

Otsu's Method was first described by Nobuyuki Otsu in 1979. This method is built on maximizing the variance $\sigma_b^2(t)$ between the classes of light and dark regions (segmented by the threshold t) named between class variance.

$$max(\sigma_b^2(t)) = max(w_f w_b(\mu_f - \mu_b)^2), \tag{1.7}$$

where w_b , and w_f are the relative proportions of background and foreground pixels in the image $(w_b + w_f = 1)$ for a given threshold value t. w_b and w_f

values calculating according to formulas:

$$w_b = \sum_{k=0}^{t-1} p_k, \quad w_f = \sum_{k=t}^{L-1} p_k, \tag{1.8}$$

where $p_k = \frac{n_k}{N}$ is probability of a pixel with intensity value k, N is the total number of pixels in the image, and L the number of grayscale values.

Mean values of μ_b and μ_f are calculated from the relations:

$$\mu_b = \frac{1}{w_b} \sum_{k=0}^{t-1} kp_k, \quad \mu_f = \frac{1}{w_f} \sum_{k=t}^{L-1} kp_k. \tag{1.9}$$

The optimal threshold value is found by a complete search of the space of all values by successively calculating the criterion values for each threshold value. The optimum threshold value is the value where the dispersion of light and dark intensities areas is the greatest - the areas do not differ much from each other, they are the most contrasting to each other.

a) Greyscale Image b) Binary Image c) Histogram

Figure 1.3: Otsu's Thresholding.

Consider an 8×8 pixel image with 8 levels of grayscale. The number of pixels n_k for each level k is given in the table below. The goal is to find the optimal threshold using Otsu's method.

k	n_k	p_k	w_b	w_f	kp_k	$\sum kp_k$	μ_b	μ_f	$\sigma_b^2(t)$
0	4	0.062	0.062	0.937	0.000	0.000	0.000	4.100	0.984
1	8	0.125	0.187	0.815	0.150	0.125	0.666	4.576	2.329
2	10	0.152	0.347	0.656	0.315	0.437	1.272	5.190	3.462
3	9	0.140	0.484	0.515	0.421	0.859	1.774	5.787	**4.023**
4	4	0.062	0.546	0.453	0.250	1.109	2.028	6.034	3.976
5	8	0.125	0.671	0.328	0.625	1.734	2.581	6.428	3.262
6	12	0.187	0.859	0.140	1.125	2.859	3.327	7.000	1.630
7	9	0.140	1.000	0.000	0.984	3.843	3.843	-	-

Table 1.1: Otsu's calculating table.

Maximum criterion value $\sigma_b^2(t)$ is 4.023, which corresponds to $k = 3$, respectively $t - 1 = 3$, i.e. the optimal threshold is $t = 4$.

Pictures 1.3 below shows the progress of the algorithm.

1.3.4 U-Net

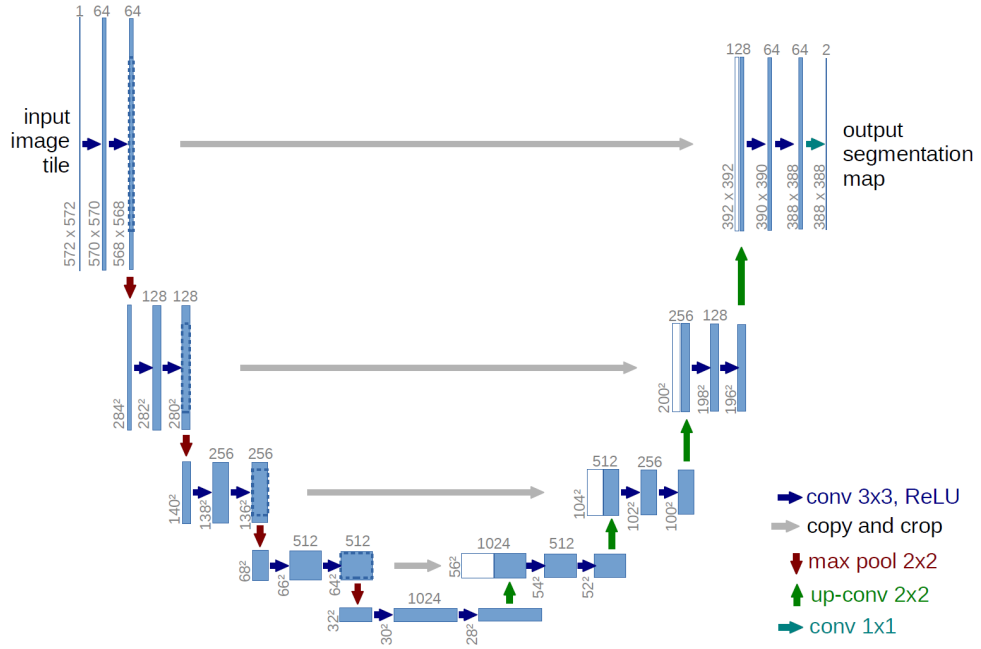


Figure 1.4: U-net architecture.

Another way of image segmentation is U-Net segmentation. U-Net is a convolutional neural network that was created in 2015 to segment biomedical images at the Department of Computer Science at the University of Freiburg [11]. Undefined terms will be explained further in this thesis (in sec. 2). The network architecture is a fully-connected convolutional network [12], modified so that it can handle fewer examples (training images) and do more accurate segmentation.

The network contains a convolutional (sec. 2.8) (left) and a upsampling part (right), so the architecture looks like the letter "U", which is reflected in the name. At each step, we double the number of feature channels.

The convolutional part is similar to a regular convolutional network, containing two consecutive 3×3 convolutional layers, followed by a ReLU (activation function described in sec. 2.2) and a 2×2 pooling function (sec. 2.9) with a step of 2.

Each step of the upsampling part contains a layer, the reverse of the pooling, which expands the feature map, followed by a 2×2 convolution, which reduces

the number of feature channels. This is followed by a concatenation with an appropriately trimmed feature map from the squeezing path and two 3×3 convolutions, each followed by a ReLU. The cropping is necessary because we lose border pixels in each convolution. On the last layer, a 1×1 convolution is used to bring each 64-component feature vector to the required number of classes. The network has a total of 23 convolutional layers.

1.4 Mathematical morphology

Mathematical morphology refers to a grouping of techniques used for image processing based on the geometric shapes of objects and structures. Morphological operations were originally defined for use on binary images, but were later extended to grayscale images and then to color images. The structuring element is a function with two variables that defines how the pixel values around the pixel being processed are considered in the calculation. Subsequently, the value of the structuring element is either added or subtracted from the pixel depending on the specific morphological operation.

1.4.1 Translation

Suppose that A is a set of pixels in a binary image, and $w = (x, y)$ is a particular coordinate point. Then A_x is the set A "translated" in the (x, y) direction.

$$A_w = \{(a, b) + (x, y) : (a, b) \in A\}. \tag{1.10}$$

In image 1.5 A is a cruciform set and $w = (2, 2)$. The set A has been shifted in the x and y directions by the values given in w. Matrix coordinates were used, instead of Cartesian coordinates, so the origin of the coordinates is at the top left, x goes down, and y goes across.

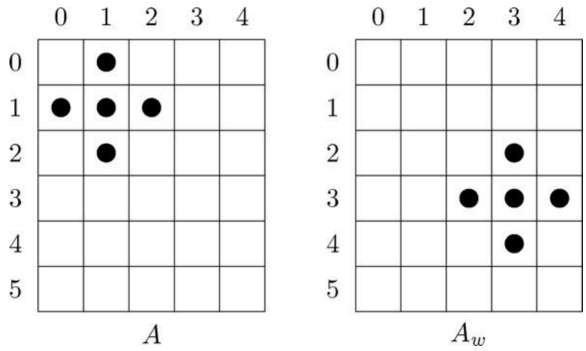


Figure 1.5: Translation illustration.

1.4.2 Dilation

Suppose A and B are sets of pixels. Then the dilation of A by B, denoted $A \bigoplus B$, is defined as:

$$A \oplus B = \bigcup_{w \in B} A_w, \tag{1.11}$$

that means is that for every point $w \in B$, we translate A by those coordinates. Then we take the union of all these translations. An equivalent definition is that

$$A \oplus B = \{(x, y) + (u, v) : (x, y) \in A, (u, v) \in B\}. \tag{1.12}$$

From this last definition, dilation is also shown to be commutative, that is

$$A \oplus B = B \oplus A. \tag{1.13}$$

An example of a dilation is given in image 1.6. In the translation diagrams, the gray squares show the original position of the object. Note that $A_{(0,0)}$ is of course just A itself. In this example, we have denoting coordinates $B = \{(0, 0), (1, 1), (-1, 1), (1, -1), (-1, -1)\}$. These are the coordinates by which we translate A. In the general case $A \oplus B$ can be obtained by replacing each point (x, y) in A with a copy of B and place the point $(0, 0)$ of B in (x, y). Equivalently, we can replace each point (u, v) of B with a copy of A.

Dilation has the effect of increasing the size of an object.

1.4.3 Erosion

Given sets A and B, the erosion of A by B, written $A \ominus B$, is defined as:

$$A \ominus B = \{w : B_w \subseteq A\}. \tag{1.14}$$

The erosion of A through B consists of all points $w = (x, y)$ for which B is in A. To perform an erosion, we can move B through A and find all the places where it will fit, and for each such place mark the corresponding $(0, 0)$ point of B. The sum of all such points forms the erosion.

In the image 1.7 can be seen the progress of the algorithm.

1.4.4 Closing

This operation may be understood as "second level" operation in that it build on basic dilation operation followed by erosion, which denoted as $A\bullet$ B:

$$A \bullet B = (A \oplus B) \ominus B. \tag{1.15}$$

An example of closing can be seen on image 1.8

Closing tends to smooth an image, but it eliminates small holes, fuses narrow breaks and thin gulfs.

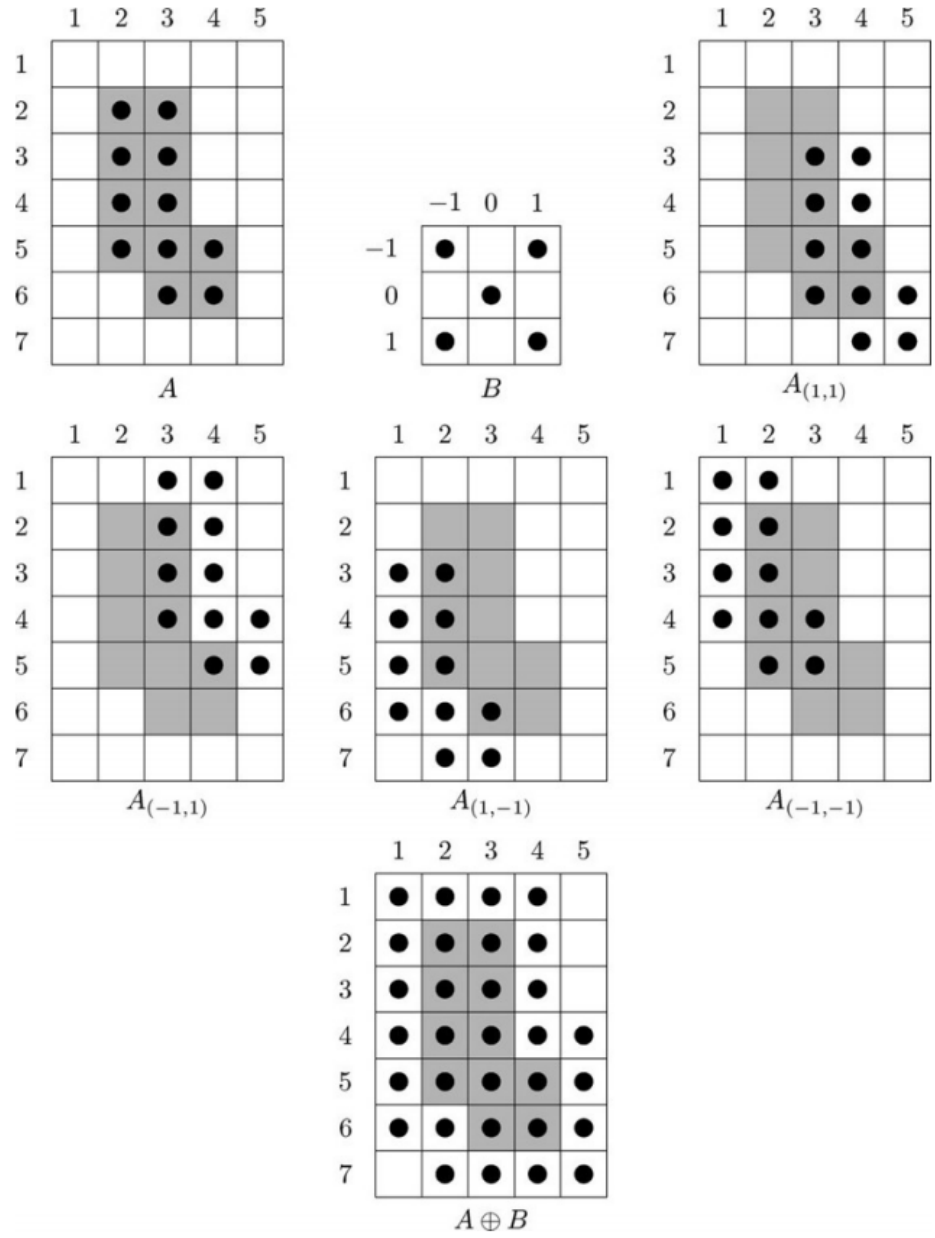


Figure 1.6: Dilation explanation.

1.4.5 Opening

Analogous to closing we can define opening. Given A and a structuring element B, the opening of A by B, denoted $A \circ B$, is defined as:

$$A \circ B = (A \ominus B) \oplus B. \tag{1.16}$$

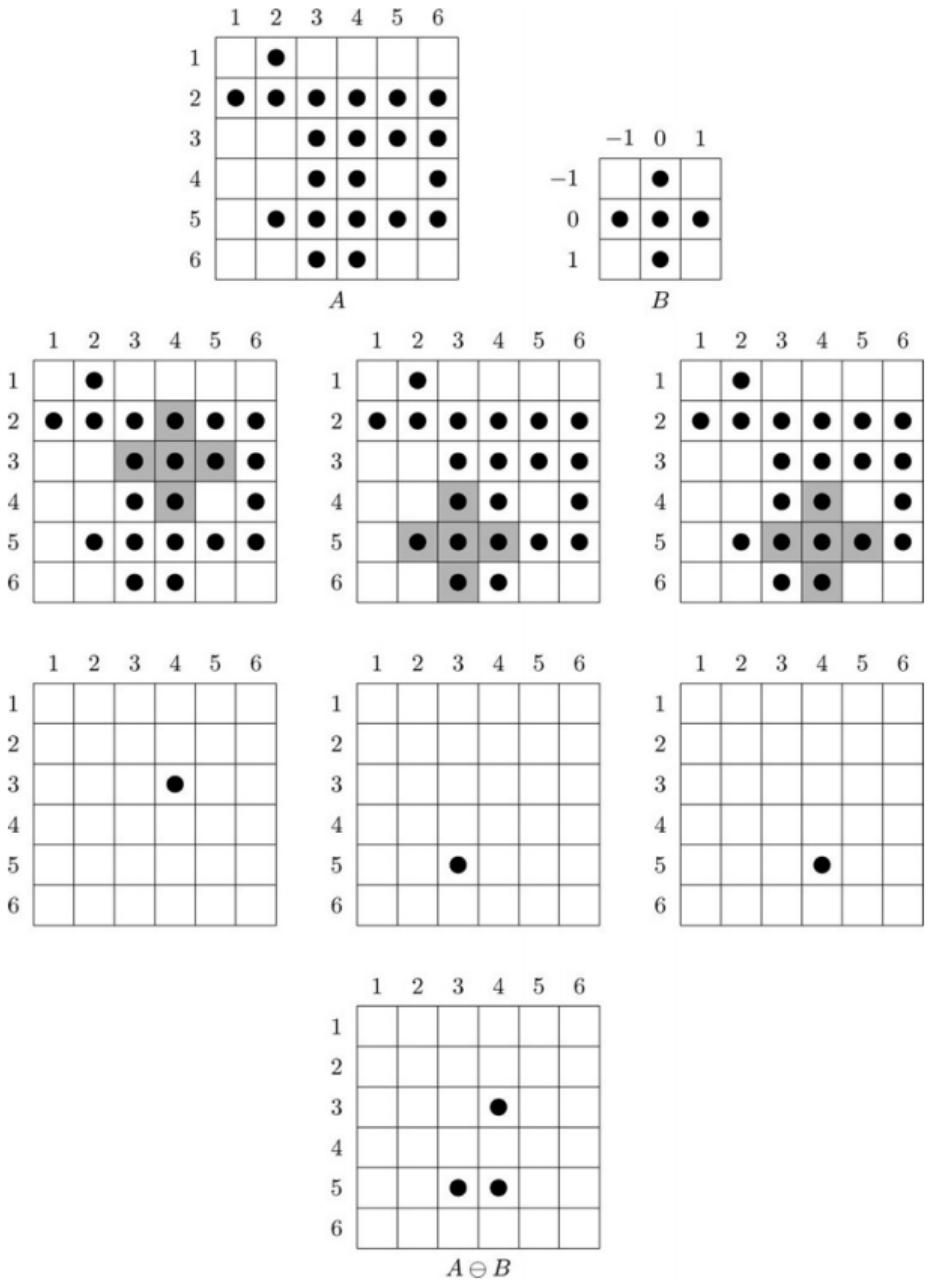


Figure 1.7: Erosion explanation with a cross-shaped structuring element.

So, an opening consists of an erosion followed by a dilation. An equivalent definition is:

$$A \circ B = \cup \{B_w : B_w \subseteq A\}, \tag{1.17}$$

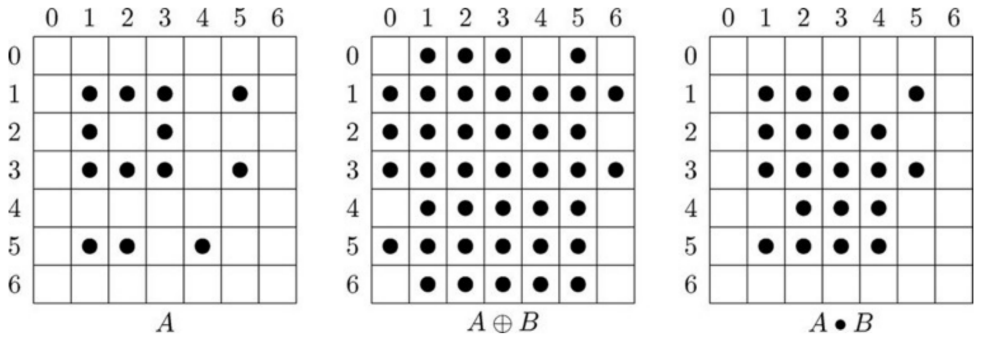


Figure 1.8: Closing explanation.

which mean that $A \circ B$ is the union of all translations of B that are placed inside A. The difference with erosion is that erosion consists only of $(0, 0)$ point of B for those translations that fit inside A. Opening consists of all of B. Here 1.9 is an example of opening.

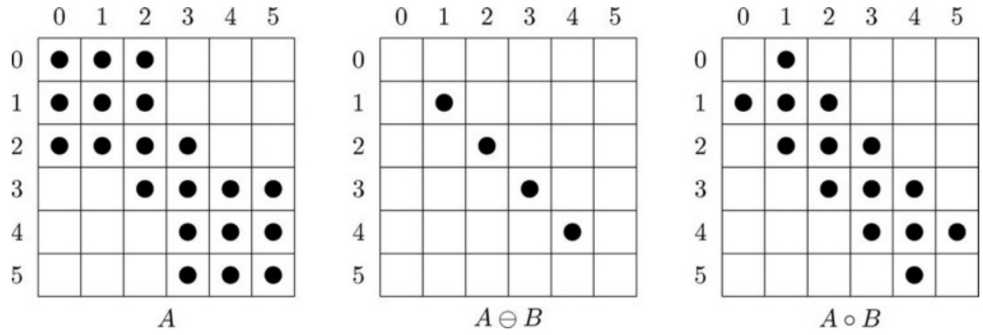


Figure 1.9: Opening explanation with a cross-shaped structuring element.

Opening tends to smooth an image, remove thin protrusions, break narrow joins.

1.5 Feature extracting

Feature extraction is part of a dimensionality reduction process in which the raw data is separated and reduced to more manageable groups. That way, when we want to process the data, it will be easier.

1.5.1 Edge detection

Edge detection is a term in image processing theory and computer vision, partly from the field of object search and object selection, based on algorithms that highlight points in a digital image where brightness changes sharply or where there are other types of inhomogeneities.

1.5.2 Gradient

Gradient is a vector of partial derivatives. It points in the direction of the largest increase in values.

$$\nabla f(x, y, ...) = \left(\frac{df}{dx}, \frac{df}{dy}, ... \right). \tag{1.18}$$

In our case we find the gradient of the image i.e. a function of two variables $f(x, y)$. The gradient modulus determines the steepness of the greatest slope or rise of the surface. The edge strength is given by the magnitude:

$$|\nabla f| = \sqrt{\left(\frac{df}{dx} \right)^2 + \left(\frac{df}{dy} \right)^2}. \tag{1.19}$$

1.5.3 Prewitt operator

In the discrete world, derivatives are approximated by differences $f(x+1) - f(x-1)$, that can be described as a linear filter masks, such as Prewitt operator:

$$P_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad P_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}, \tag{1.20}$$

where P_x is operator for vertical and P_y is for horizontal edges.

1.5.4 Sobel operator

The Sobel operator is based on image convolution by small separable integer filters in the vertical and horizontal directions, so it is relatively easy to compute. On the other hand, the gradient approximation used by it is rather coarse, especially it affects the high-frequency oscillations of the image. However, it is good enough for practical application in many problems. More precisely, the operator uses intensity values only in the vicinity 3×3 of each pixel to obtain an approximation of the corresponding image gradient, and uses only integer values of brightness weights to estimate the gradient.

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \tag{1.21}$$

At each point of the image the approximate value of the gradient value can be calculated by

$$S = \sqrt{S_x^2 + S_y^2}. \tag{1.22}$$

We can also calculate the direction of the gradient:

$$\Theta = arctan\left(\frac{S_y}{S_x}\right). \tag{1.23}$$

Below 1.10 is a demonstration of how the filters work.

a) Original image b) Prewitt c) Sobel

Figure 1.10: Operators work illustration.

1.5.5 Non-maximums suppression

The goal of non-maximums suppression is to keep only those pixels that belong to the edge. The way to do it is to assign to each pixel a direction θ described above called edge gradient. The pixel that forms the edge must then have a larger intensity than its local neighbours in the direction of the edge gradient of that pixel. The computed edge gradient does not point to a specific pixel, but between pixels, so it is we need to consider the extent of their contribution to the edge gradient calculation based on neighbouring pixels. We take the neighbouring pixels of the gradient as the ones between which the gradient is directed and calculate the weighted average from them. For example, in the figure 1.11, this would be the two red highlighted pixels on the top right.

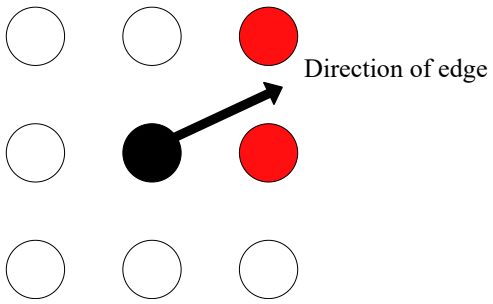


Figure 1.11: Illustration of finding of edge direction.

1.5.6 Hysteresis thresholding

Hysteresis thresholding is a method that uses two thresholds: a lower threshold t_l and an upper threshold t_u. A pixel which value is greater than t_h is considered as a strong edge pixel. A pixel which value is between t_l and t_h, and which is at the same time adjacent to another edge pixel (strong edge pixel), is considered a weak edge pixel.

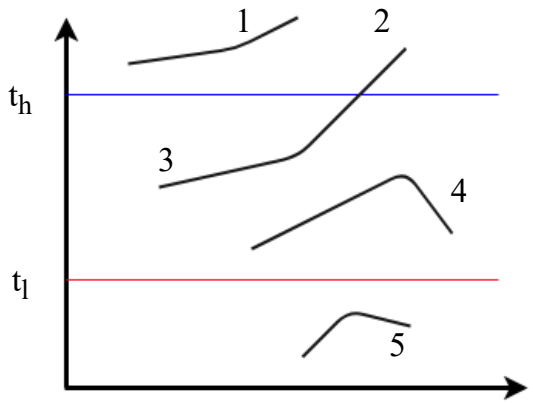


Figure 1.12: Hysteresis thresholding.

In figure 1.12, 1 and 2 are strong edges as they are above high threshold t_h. Similarly, 5 is a sure non edge. Both 4 and 3 are weak edges but since 3 is connected to 2 which is a sure edge, 3 is also considered as a strong edge. Using the same logic 4 is discarded. This way we will get only the strong edges in the image.

1.5.7 Canny edge detector

The Canny detector [13] in the discipline of computer vision is an image border detection operator. It was developed in 1986 by John F. Canny and uses a multi-step algorithm to detect a wide range of boundaries in images.

Canny studied the mathematical problem of obtaining a filter optimal according to the criteria of selection, localization, and minimization of several responses of one edge. He showed that the desired filter is the sum of four exponents. He also showed that this filter could be well approximated by the first derivative of the Gaussian. Canny introduced the notion of Non-Maximum Suppression, which means that the pixels of the edges are declared to be the pixels where the local maximum of the gradient in the direction of the gradient vector is reached.

The steps of the algorithm are as follows:

- smoothing (Gaussian filter for example),

- finding gradients (Sobel filter for example),
- non-maximums suppression,
- dual threshold filtering (hysteresis thresholding),

Chapter 2

Soft computing methods

Soft computing is a concept introduced by Lotfi Zadeh in 1994 [14], uniting into a general class imprecise, approximate methods for solving problems, often with no solution in polynomial time.

Soft computing technologies are oriented to solving control problems with weakly structured control objects [15]; soft computing tools often use the technique of artificial neural networks [16]. Various soft computing methods can complement each other and are often used together, forming hybrid systems [17]. Referring to experiments published in article [18], the most successful OCR can only be done using machine learning algorithms.

2.1 Artificial neuron

An artificial neuron is a node of an artificial neural network, which is a simplified model of a natural neuron. Mathematically, a neuron is a weighted adder, whose single output is defined through its inputs and matrix of weights as follows:

$$y = f(u), \quad u = \sum_{i=1}^{n} w_i x_i + w_0, \tag{2.1}$$

where x_i and w_i are the signals at the neuron inputs and the weights of the inputs respectively, the function u is called the induced local field, and $f(u)$ is an activation function. Possible values of signals at the neuron inputs are considered to be given in the interval $[0, 1]$.

2.2 Activation function

Activation function $f(u)$ defines the dependence of the signal at the output of the neuron on the weighted sum of the signals at its inputs. In most cases it is monotonically increasing and has a value range of $[-1, 1]$ or $[0, 1]$, but

there are exceptions. Also, for some learning algorithms it is necessary for the network to be continuously differentiable on the entire numerical axis.

An example of the activation function is *Rectified linear unit* (ReLU).

$$f(u) = \begin{cases} 0, & u < 0 \\ u, & u \geq 0 \end{cases} \tag{2.2}$$

2.3 Loss function

In statistical decision theory loss function is a function that characterizes the losses in incorrect decision-making based on observed data. If the problem of estimation of a signal parameter against the background of noise is solved, the loss function is a measure of the discrepancy between the true value of the estimated parameter and the estimate of the parameter.

2.4 Cross-entropy

The cross-entropy or logarithmic loss function measures the difference between two probability distributions. If the cross-entropy is large, it means that the difference between the two distributions is large, and if the cross-entropy is small, the distributions are similar to each other.

In the case of binary classification, the formula is as follows:

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i log(p(y_i)) + (1 - y_i) log(1 - p(y_i)), \tag{2.3}$$

where y is class label, $p(y)$ is the predicted probability and N is amount of samples.

2.5 Intersection over Union metric

Intersection over Union (IoU) metric, also known as the Jaccard index, is a number from 0 to 1 that shows how much the inside area of two objects (the reference *ground truth* and the current object) overlap. Formally, for two non-empty sets A and B , the function IoU is defined as:

$$IoU(A, B) = \frac{|A \cap B|}{|A \cup B|}. \tag{2.4}$$

2.6 Backpropagation

Training the neural network (adjusting the weights) is called backpropagation and uses a gradient method called stochastic gradient descent. The gradient

of the activation function specifies the direction (represented by a sign) and magnitude by which the corresponding weight should be changed to achieve more optimal value in a given iteration. Thus, instead of immediately searching for the best solution, the goal is reaching by successive steps. If the model is working correctly, the gradient should converge to zero.

Weights adjust after each training example and thus "move" in the multidimensional space of weights. To "get" to the minimum error, we need to "move" in the opposite direction to the gradient, that is add to each weight $w_{i,j}$ gradient

$$\nabla w_{i,j} = -\eta \frac{dE}{dw_{i,j}}, \tag{2.5}$$

where $0 < \eta < 1$ is a multiplier specifying the speed of "movement" called *learning rate*.

2.7 Convolution

Convolution is an operation with a pair of matrices A (of size $n_x \times n_y$) and B (of size $m_x \times m_y$), the result of which is matrix $C = A * B$ of size $(n_x - m_x + 1) \times (n_y - m_y + 1)$. Each element of the result is calculated as the scalar product of matrix B and submatrix A of the same size (the submatrix is determined by the position of the element in the result). That is

$$C_{i,j} = \sum_{u=0}^{m_x-1} \sum_{v=0}^{m_y-1} A_{i+u,j+v} B_{u,v}. \tag{2.6}$$

In 2.1 can be seen how matrix B "moves" over matrix A, and at each position the scalar product of matrix B and the part of matrix A on which it is now superimposed is counted. The resulting number is written to the corresponding element of the result.

2.8 Convolutional layer

The convolution layer of the neural network is an application of the convolution operation to the outputs from the previous layer, where the convolution kernel weights are trainable parameters. Another trainable weight is used as a constant shift (bias).

One convolution layer may contain several convolutions. In this case, for each convolution, the output will have a different image. For example, if the input has dimension $w \times h$, and the layer has n convolutions with $k_x \times k_y$ kernel, then the output will have dimension $n \times (w - k_x + 1) \times (h - k_y + 1)$;

The convolution kernels may be three-dimensional. The convolution of a three-dimensional input with a three-dimensional kernel occurs similarly. The

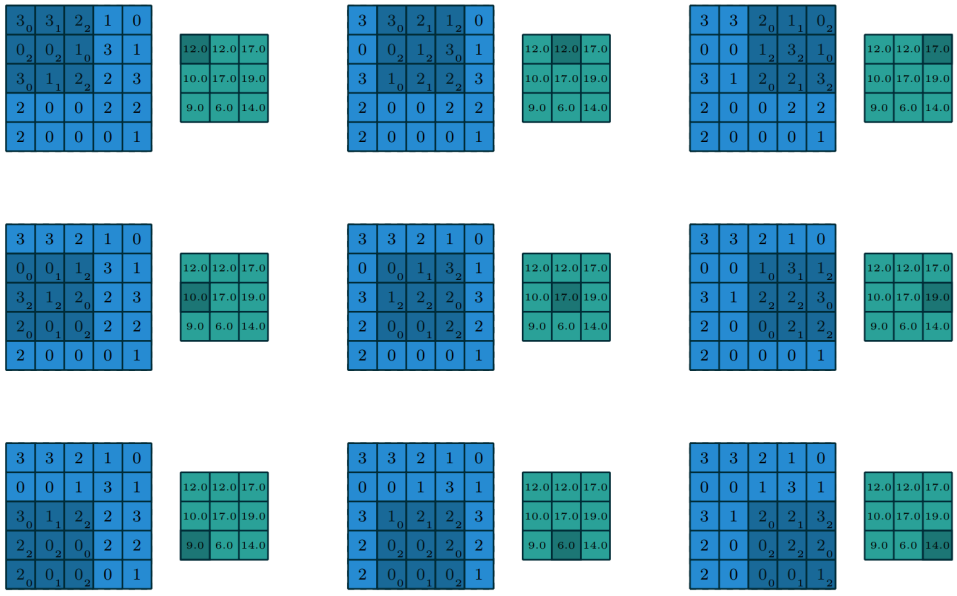


Figure 2.1: Example of convolution of two 5×5 and 3×3 matrices [1].

scalar product is also counted over all image layers. For example, to average color information of the original image, a convolution of dimension $3 \times w \times h$ can be used on the first layer. The output of such a layer will be one image (instead of three); Here 2.2 is an example.

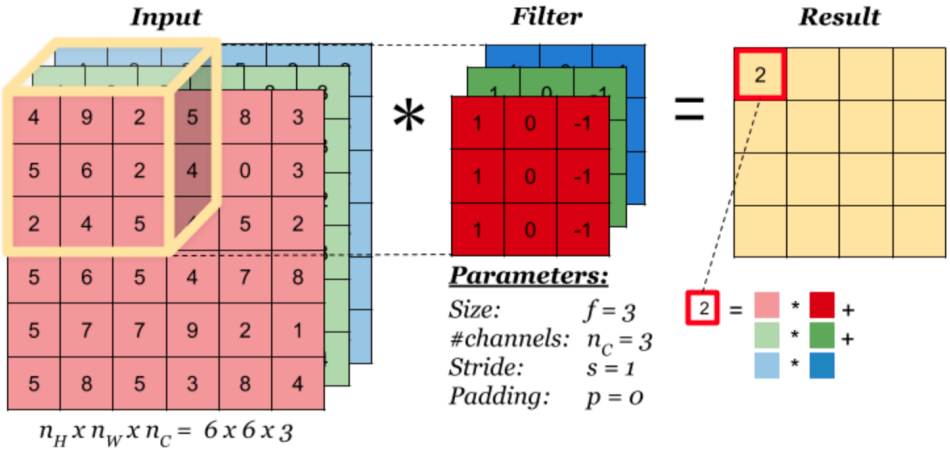


Figure 2.2: A convolution with a three-dimensional kernel [2].

The convolution operation shrinks the image. Also pixels that are on the edge of the image participate in fewer convolutions than internal ones. For this reason, image *padding* is used in convolution layers. Outputs from the

previous layer are augmented with pixels, so that the image size is retained after convolution. Such convolutions are called same convolution, and convolutions without image augmentation are called valid convolution.

Another parameter of the convolution layer is the *stride*. Although convolution is usually applied in a row for each pixel, sometimes a shift other than one is used. The scalar product is not counted with all possible kernel positions, but only with positions multiples of some shift s. If input has dimension $w \times h$, and convolution kernel has dimension $k_x \times k_y$ and shift s is used, then output will have dimension $\lfloor \frac{w-k_x}{s} + 1 \rfloor \times \lfloor \frac{h-k_y}{s} + 1 \rfloor$.

2.9 Pooling layer

The pooling layer is designed to reduce the dimensionality of the image. The initial image is divided into blocks of size $w \times h$, and for each block some function is calculated. The most often used function is max pooling or (weighted) average pooling. This layer has no teachable parameters.

How pooling works can be seen on a figure 2.3.

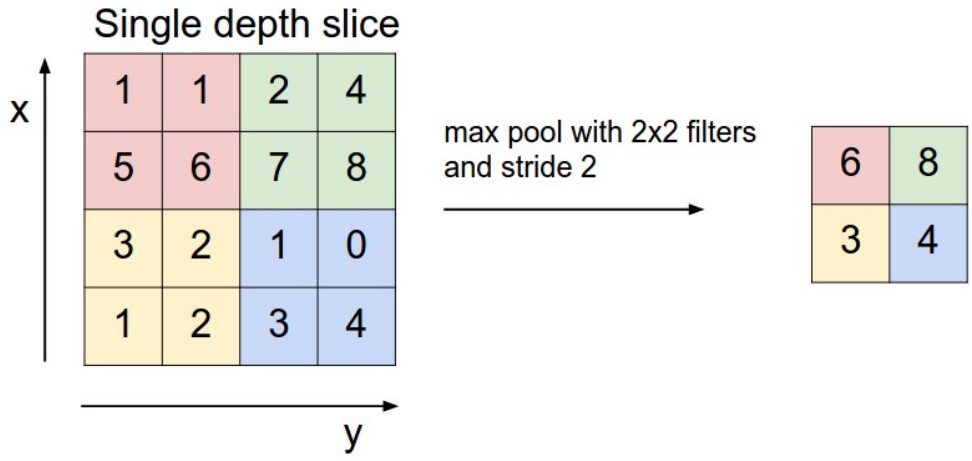


Figure 2.3: Example of a pooling operation with a maximum function [2].

The main purposes of the pooling layer:

- image reduction, so that next convolutions could operate on a larger area of the original image;
- increasing the invariance of the network output with respect to small input transfer;
- speeding up calculations.

2.10 Softmax nonlinearity

For classification, it is necessary to create a layer in which each neuron is connected to each other. Such layer is called FCL (*Fully Connected Layer*). Although classification based only on convolutional and pooling layers would also be possible, the creation of a fully connected layer a more convenient and relatively inexpensive way to introduce additional nonlinearity into the model.

When the number of possible classes is more than two, the Softmax function is used. The function converts a vector z of dimension K to a vector σ of the same dimension, where each coordinate σ_i of the resulting vector is represented by a real number in the interval $[0, 1]$ and the sum of coordinates is equal to 1.

The coordinates σ_i are calculated as follows:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{k-1}^{K} e^{z_k}} \tag{2.7}$$

The coordinates σ_i of the resulting vector are interpreted as probabilities that the object belongs to class i.

2.11 Recurrent neural network

Diagrams and main thoughts mainly follow this article [19].

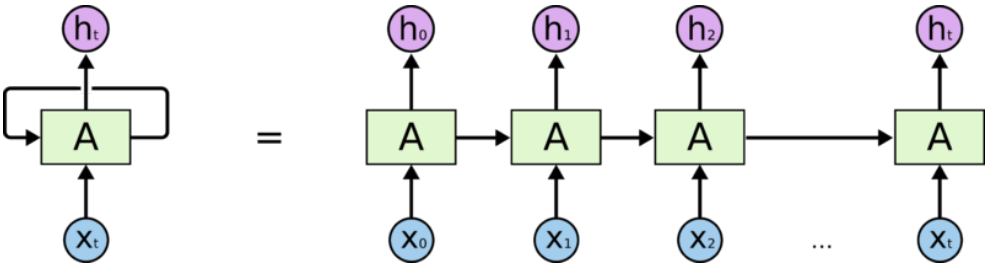


Figure 2.4: RNN and its expanded representation.

Recurrent neural networks are networks with loops that are well suited for processing sequences. RNN training is similar to training an ordinary neural network. We also use the backpropagation algorithm, but with a slight modification. Since the same parameters are used at all temporal steps in the network, the gradient at each output depends not only on the computation of the current step, but also on the previous temporal steps. For example, to calculate the gradient for the fourth element of the sequence, we would have to "propagate the error" to 3 steps and sum the gradients. This algorithm is called *Backpropagation Through Time* (BPTT) algorithm [20].

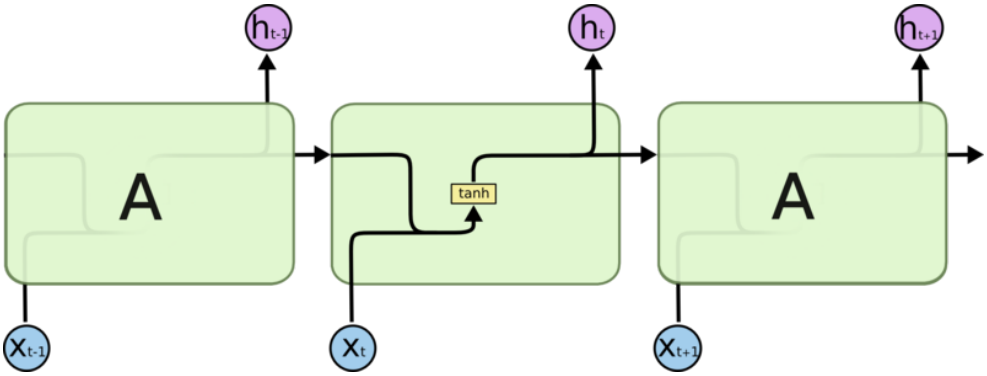


Figure 2.5: Diagram of the recurrent network layer.

2.12 Long short-term memory

Long-term memory (LSTM) is a special kind of recurrent neural network architecture capable of learning long-term dependencies, proposed in 1997 by Sepp Hochreiter and Jürgen Schmidhuber [21]. Recurrent neural networks add memory to artificial neural networks, but the realized memory is short - at each learning step the information in the memory is mixed with the new one and after several iterations it is completely overwritten.

LSTM modules are specifically designed to avoid the problem of long-term dependency by remembering values for both short and long periods of time. This is because the LSTM module does not use an activation function within its recurrence components. Thus, the stored value does not erode over time and the gradient does not disappear when using the backward error propagation over time method when training the network.

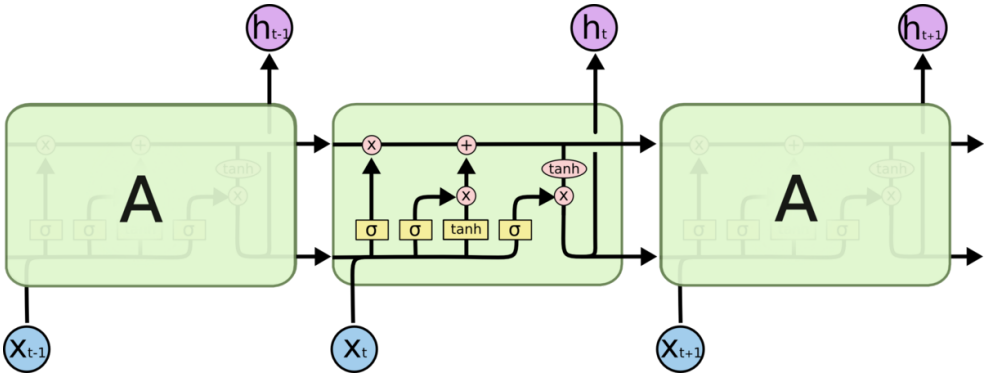
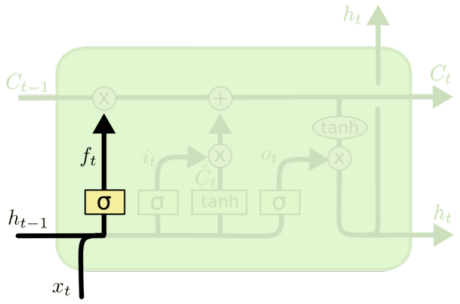


Figure 2.6: Diagram of the Long-Term Memory layers.

The key components of an LSTM module: the cell state and various filters. The cell state can be spoken of as the memory of the network, which transmits the relevant information throughout the module chain. In this way, even

information from early timesteps can be retrieved at later ones, leveling out the effect of short-term memory.

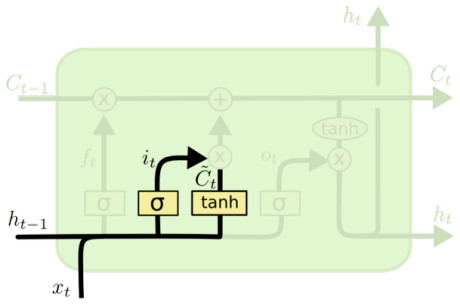
First the "forget gate layer" determines which information can be forgotten or left behind. The values of the previous output $h_t - 1$ and the current input x_t are passed through the sigmoidal layer. The resulting values are in the range $[0; 1]$. Values that are closer to 0 will be forgotten, and those closer to 1 will be left (figure 2.7).



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure 2.7: Forget gate layer.

Next, it is decided what new information will be stored in the cell state. This step consists of two parts. First, a sigmoidal layer called the input layer gate determines which values to update. Then the tanh layer [22] constructs a vector of new candidate values $\tilde{C}_t$ that can be added to the cell state (figure 2.8).

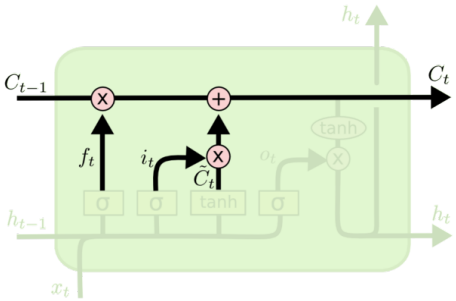


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figure 2.8: Input layer gate and tanh laye.

To replace the old cell state $C_t - 1$ with the new cell state C_t it is necessary to multiply the old state by f_t, forgetting what we decided to forget earlier. Then adds $i_t * \tilde{C}_t$. These are the new candidate values multiplied by t (how much to update each of the state values). Details illustrated at figure 2.9.

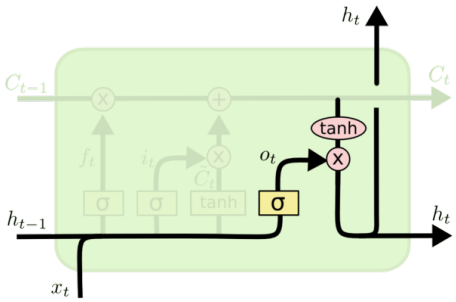
The last step determines what kind of information the output will be. The output will be based on our cell state and some filters will be applied to it. First, the values of the previous output $h_t - 1$ and the current input x_t are passed through a sigmoidal layer, which decides what information from the cell



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 2.9: Change of the main "highway".

state will be output. The cell state values are then passed through the tanh layer to get output values from the range -1 to 1 and multiplied with the output values of the sigmoidal layer, which allows only the required information to be output 2.10.



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 2.10: Change of the secondary "highway".

The h_t and C_t thus obtained are passed on down the chain.

2.13 Beam search

This algorithm will be used further in one of the best OCR systems, so it is important to describe it first. Beam search is one of the algorithms for searching the state space. It is an optimization of best-first search. It is based on the idea of ordered search to continue searching always from the most promising node, supplemented by "pruning" the least promising branches, which reduces memory requirements. For each searched node, all its successors are sorted according to a given heuristic, and only a certain number given by the so-called "beamwidth", which is fixed in the basic version of the algorithm, are then put into the priority queue for further search. When the beamwidth is set to infinity, the algorithm corresponds to an ordered search algorithm.

2.14 Transfer Learning

Transfer learning is a task in machine learning aimed at storing knowledge gained in one task and applying it to another but related task [23]. For example, knowledge gained during learning to recognize one type of text, font, can be applied when trying to recognize others. From a practical perspective, reusing or transferring information from previously trained tasks to train new tasks has the potential to significantly improve the effectiveness of a machine learning model.

Chapter 3

OCR software

Before startning to scan documents, it is important to know the features of modern software. Because of importance of text recognition, the competition in this area is quite high. This chapter will survey the currently available software. The software divided into categories of commercial and non-commercial use. Commercial software defined as software that is provided under a paid license in the form of a subscription. Non-commercial software is software that is licensed under one of the open source licenses or just free to use.

3.1 Commercial

3.1.1 ABBYY FineReader

ABBYY FineReader 3.1 is an optical character recognition program developed by the Russian company ABBYY. ABBYY FineReader PDF 15 makes it easy to digitize, search, convert, edit documents, share files and work together on any type of documents. The program allows to translate images of documents into electronic editable formats. In particular into Microsoft Word, Microsoft Excel, Microsoft Powerpoint, Rich Text Format, HTML, PDF/A, searchable PDF, CSV. The program supports text recognition in 192 languages and has a built-in spell checker for 48 of them. There are more than 20 million ABBYY FineReader users in the world. FineReader is based on ABBYY OCR optical character recognition technology licensed by Fujitsu, Panasonic, Xerox, Samsung and others.

3.1.2 OmniPage

OmniPage Ultimate 3.2 is professional software for converting images (JPG and PNG), documents, and PDF files into digital files. Available from Kofax Incorporated. OmniPage can accurately digitize images and documents, making them both editable and searchable. It also supports a long list of image formats,

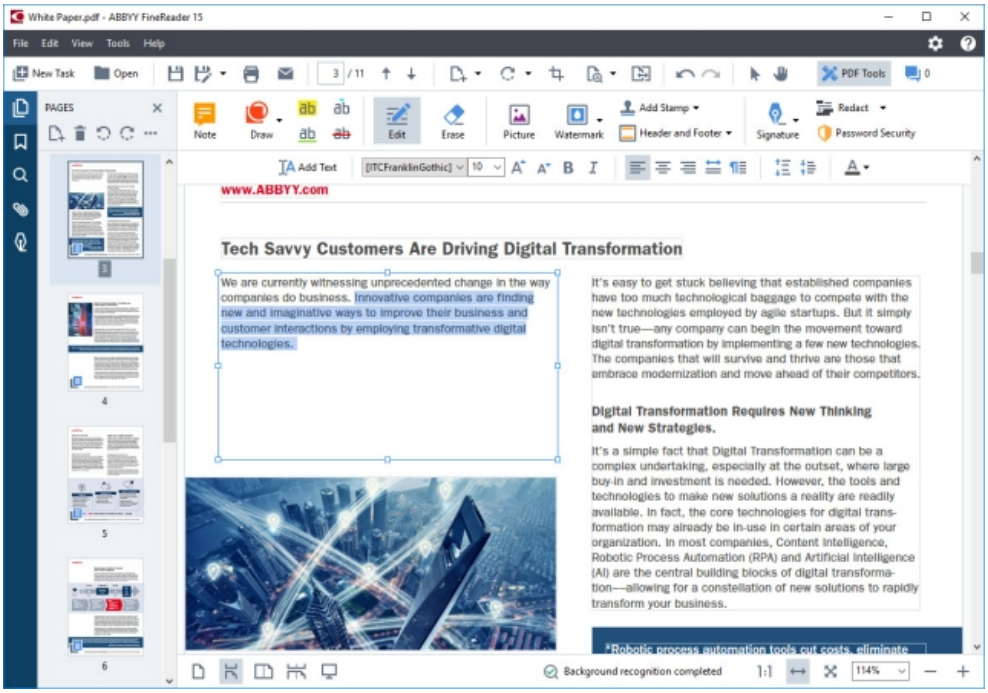


Figure 3.1: ABBYY FineReader program.

so regardless of the file extension you can easily convert it to the file format you want. OmniPage Ultimate uses its proprietary technology to detect the layout of images and automatically rotates the document in the correct orientation. It is also possible to work with both vertical and horizontal text.

Figure 3.2: OmniPage program.

3.1.3 Adobe Acrobat Pro DC

Adobe Acrobat 3.3 is a software package produced in 1993 by Adobe Systems and designed for creating and viewing electronic publications in PDF format. Adobe (the creator of PDF fromat and various document standards) has developed a powerful text recognition engine for accurately extracting texts from PDF files that may be represented as scanned images for example. Although it is not as feature-rich as ABBYY FineReader, Adobe Acrobat certainly excels it at extracting. For example, you can easily import text PDF files into Adobe Acrobat and then use text recognition technology to convert the file into editable text. However, if you want to process an image, you first need to create PDF file from it and only then can you work with that.

The program saves the font of the original document using the method of creating custom fonts. Since Adobe has a huge repository of branded, common and designer fonts, it automatically matches the font style of the source document and then converts the PDF to that particular font. And in case there is no font available, the program creates its own font using similar typography. This is a feature that only Adobe can use.

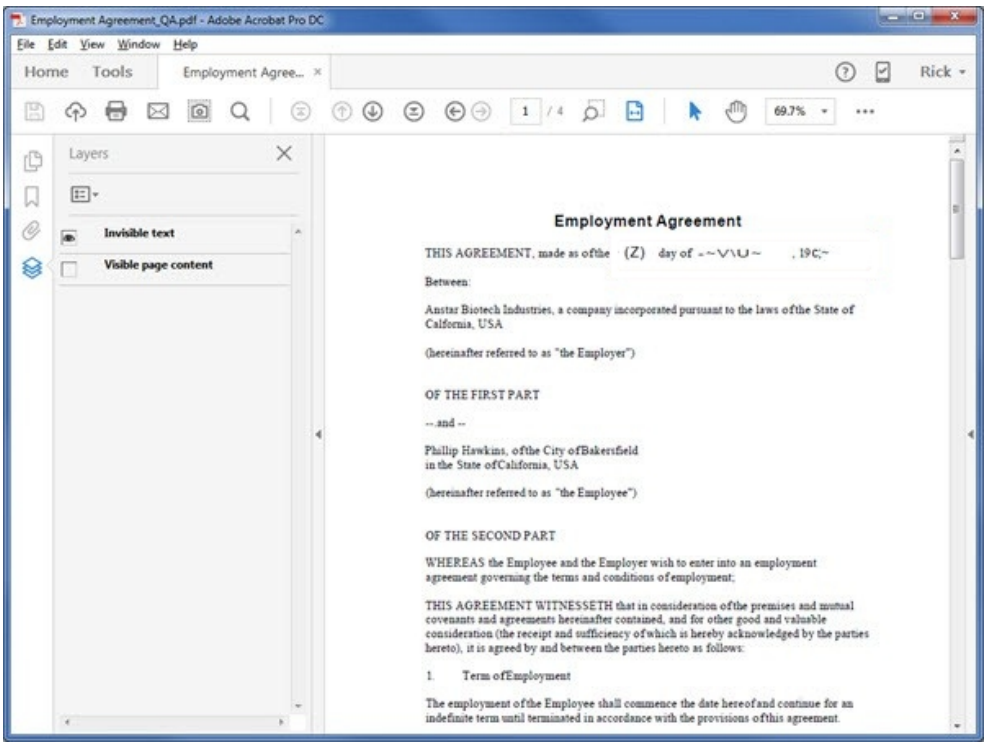


Figure 3.3: Adobe Acrobat Pro DC program.

3.2 Non-commercial

Under the conditions of the research testing was conducted on open for distribution, the so-called Open Source programs. Non-commercial software is inferior to their counterparts in the quality of recognition, the number of built-in languages, as well as the lack of user interfaces. The interaction is performed using terminal commands or through frameworks (external libraries). User also often has to pre-process the image on their own to achieve better results.

3.2.1 Tesseract OCR

According to Google trends statistics (figure 3.4), Tesseract is considered to be the most popular OCR engine of the last 5 years, so it was used as a benchmark.

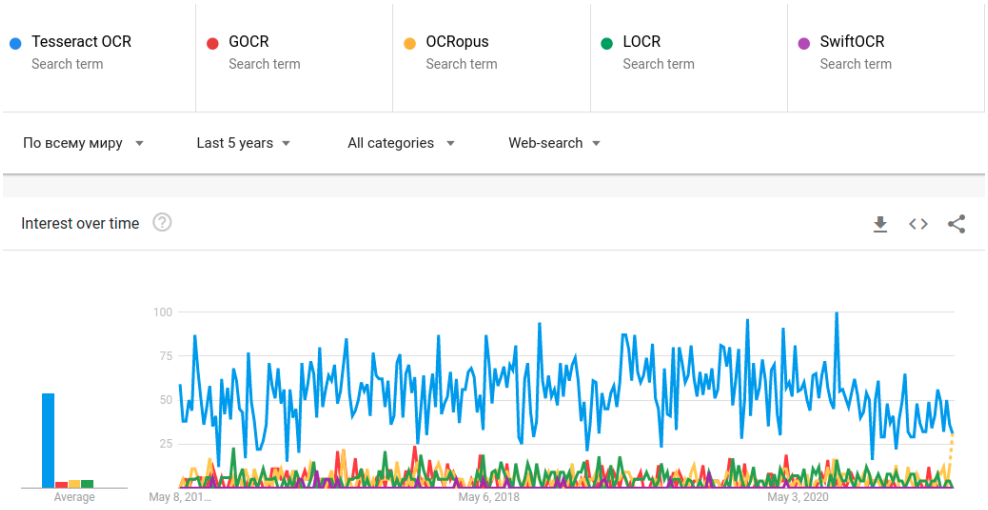


Figure 3.4: Google trends comparison for different open source OCR tools.

Tesseract is a free computer program for text recognition that was developed by Hewlett-Packard from the mid-1980s to the mid-1990s. In 2005 Hewlett-Packard released Tesseract as open source software. Since August 2006, it has been developed by Google. It is available under the Apache 2.0 license. At the moment the program already works with UTF-8. Languages support is provided by additional modules (more than 192). It also supports a variety of image formats, different types of recognition (image as a word, block of text, vertical text) and easy customization.

3.2.1.1 Architecture

Tesseract takes binary images with optional text regions as input. Recognition is done mostly in the traditional way, but with some modifications. Explanation follows analysis by Ray Smith [4]. Text lines are broken into words in different ways depending on the type of character spacing. Tesseract searches for patterns in pixels, letters, words, and sentences, using a two-step approach which includes adaptive classifier.

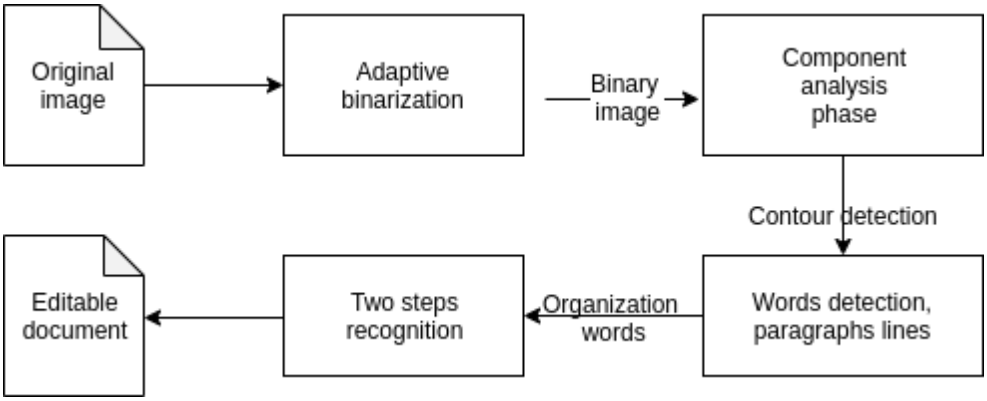


Figure 3.5: Tesseract OCR Architecture [3].

It takes one pass through the data to recognize the characters. Each satisfactory word is passed to the adaptive classifier as training data. It allows better recognition of the text that goes next. However, useful information for training may be at the very end of the text (at the bottom of the page). So second pass is made to recognize the least successful recognition cases during the first phase. Tesseract replaces them with the letters most likely to match the word or context of the sentence.

3.2.1.2 Line finding

Lines are searched by filtration of so-called blobs and line construction. Blobs are formed by the contours of characters. After analyzing the page layout for text blocks (whether they are in columns, forming separate chunks), Tesseract removes drop-down lines, vertically contiguous characters, as well as noise with a median filter and percentile height filter [24]. Blobs are then combined into rows based on their y-axis coordinates as well as heuristic criterias.

3.2.1.3 Baseline fitting

The lines are then fitted more precisely with the square spline [25]. This technique allows you to process text with curved baselines [26]. This artifact often appears during scanning. The algorithm knocks down groups of characters, with a fairly uncontinuous offset relative to the original straight baseline. The

quadratic spline is then guided to the most populous group of symbols, which is the baseline, using the method of least squares. Result of this process is shown in 3.6.

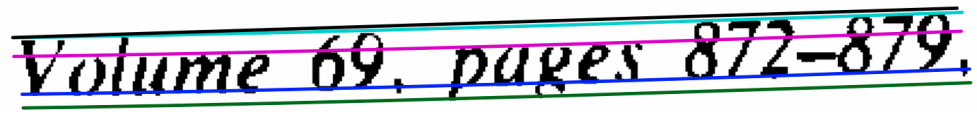


Figure 3.6: An example of a curved fitted baseline [4].

Baseline shown as bottom blue line. Purple line is named as mean line.

3.2.1.4 Character chopping

After the characters have formed into words, there is a possibility that there are no gaps between them. Some characters stick together when they are printed poorly. Tesseract tries to find such cases and splits words into characters using a certain constant pitch. In figure 3.7 is an example.

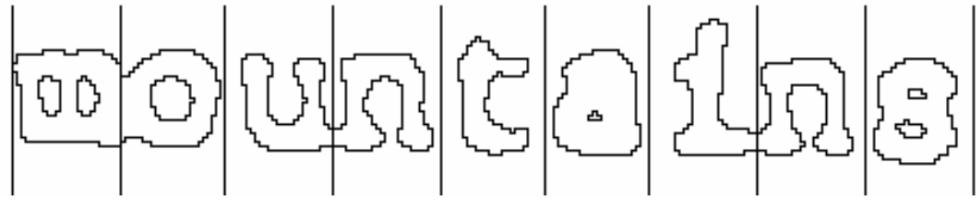


Figure 3.7: A fixed-pitch chopped word [4].

In case of text non-fixed pitch (figure 3.8), Tesseract measures the gaps between the baseline and mean line. Gaps close to the threshold become fuzzy, so the final decision may change after the word is recognized.

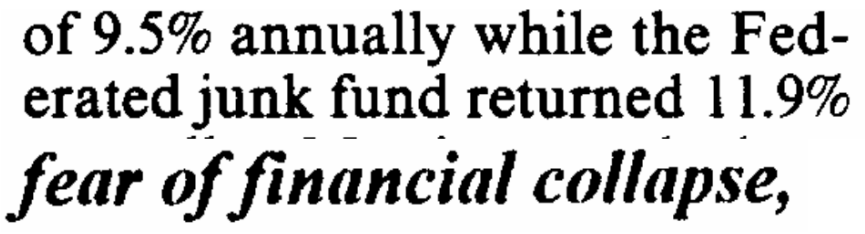


Figure 3.8: Some difficult word spacing of non-fxed pitch text [4].

3.2.1.5 Word recognition

Next, the algorithm shreds the blobs. In case of unsatisfactory result at the previous step, the tesseract tries to shrink the blob with the least confidence

from the classifier. The points that could potentially become candidates are in the concave vertices of the polygonal approximation [27]. An example is given in figure 3.9.

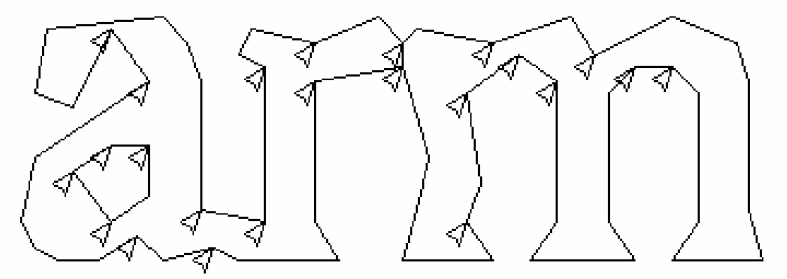


Figure 3.9: Candidate chop points and chop [4].

If the word is still not good enough, it lends itself to further processing. The so-called associator searches the segmentation graph for possible combinations of maximally chopped blocks (parts of letters) using the algorithm A* [28]. An example of such chunks is shown in figure 3.10.

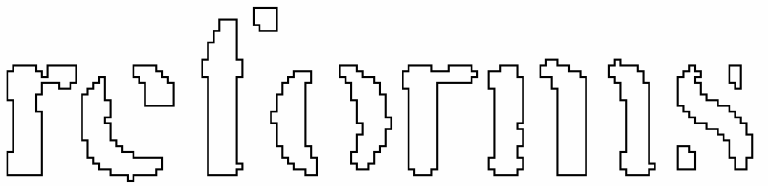


Figure 3.10: Associating broken characters [4].

3.2.1.6 Character classification

At first, Tesseract used a static character classifier based on topological features. However, this method handled broken letters poorly (figure 3.11).

The idea was to use polygonal approximation segments as features, but this approach is also not robust to broken characters. The features of the unknown data do not necessarily have to match the features of the training data. This problem has been solved by extracting many small features of varying length from the unknowns and combining them to match more robust symbol characteristics. The process of matching small features to large features showed good results. But the computational cost of doing so is quite high.

The classification itself is done by matching the symbol with instances of training data. The number of similar features is used to decide which letter to identify the processed picture to. The features are also counted using heuristics.

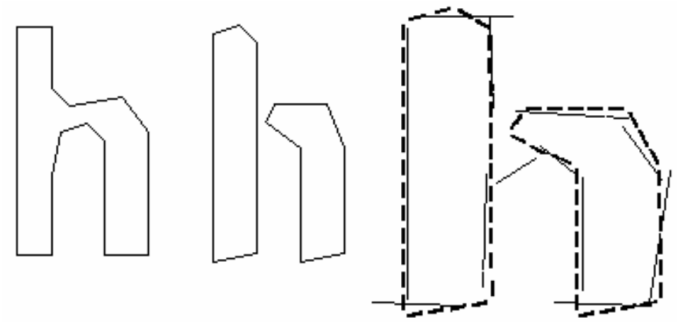


Figure 3.11: (a) Original letter 'h', (b) broken 'h', (c) features matched to prototypes [4].

3.2.1.7 LSTM Integration

In October 29, 2018, a version of the program, Tesseract 4, was released. The update made it possible to use LSTM [21] neural networks. Input image is split into blocks of text and then fed to the LSTM model.

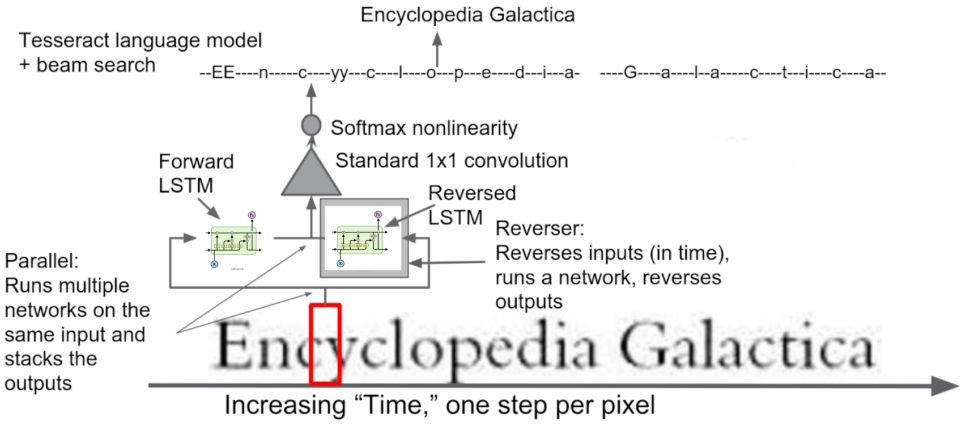


Figure 3.12: Tesseract using LSTM [5].

LSTM neural networks outperform all other alternative neural network architecture models for this type of pattern recognition. Their application is also superior to the more "classical" character recognition algorithms used in various popular commercial products. For example, the LSTM network has achieved the best known results in unsegmented coherent handwriting recognition, and won the ICDAR handwriting recognition competition in 2009. If the Tesseract recognizer based on the LSTM neural network fails on a particular sequence of characters, it can "revert" to its general static shape classifier for detection. So, in essence, the Tesseract LSTM is two OCR classifiers [29].

3.2.2 GOCR

To complete the picture, it is important to consider an algorithm that uses more classical methods of text recognition.

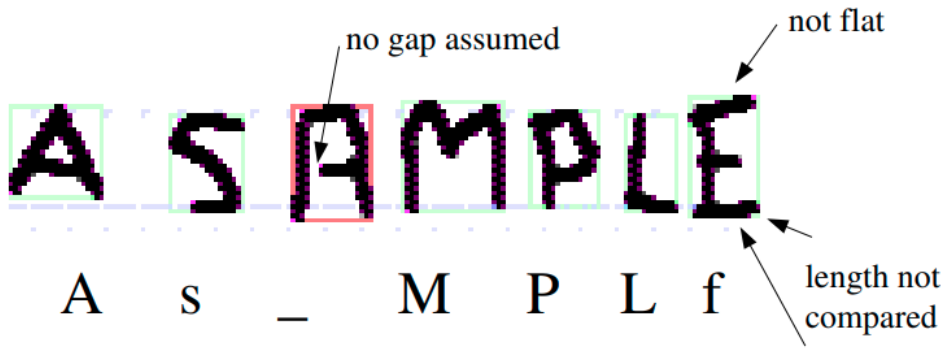


Figure 3.13: GOCR processing problems [6].

GOCR (or JOCR) [6] is a freely distributed optical character recognition program developed by Jörg Schulenburg. Can be used to convert or scan images (portable bitmaps or PCX) to text files. GOCR is capable of processing unprinted fonts 20-60 pixels high. Program has issues with character overlaps, embossed fonts, handwriting, heterogeneous fonts, noisy images, high-skewed text, and any non-Latin characters. Between versions 0.40 (March 2005) and 0.43 (December 2006), the recognition mechanism was gradually replaced by a vector version. The program was originally called GOCR which means GNU Optical Character Recognition. When it came time to register the project on SourceForge, the GOCR name was already taken, so the project was registered as JOCR (Jörg's Optical Character Recognition). As a result, the project and program are known as GOCR and JOCR. Valid formats: PBM, PGM, PPM, PNM, PCX (some), TGA.

Chapter 4

Comparison methods

4.1 Comparison metrics

This section mainly follows instructions suggested by Smitka, J. and Borkovcova, M. [18]. The algorithms are compared according to:

- quality of character recognition (error rate),
- the amount of memory that the algorithm needed to run,
- the amount of time the algorithm needed to run.

The error rate is calculated as the ratio of incorrectly recognized (or unrecognized) characters to the total number of characters in the scanned document.

The amount of memory is measured as the maximum peak of allocated memory in run-time.

The time required to run the algorithm is variable – it depends on the hardware we use. So we need to recalculate this value in such a way as to eliminate the influence of the hardware. For this purpose, we can determine a certain standard for comparison purposes. As this standard, we can use the Tesseract Open Source OCR Engine. We can run this algorithm in a 1-thread environment.

Let t_{tess} be the time required by the Tesseract algorithm to solve the OCR problem on the basic dataset of historical documents. Therefore the time score of the tested algorithm can be calculated as a time required by the tested algorithm divided by the value of t_{tess} (and multiplied by 100 to obtain the percentage value):

$$t_i = \frac{t_{alg}}{t_{tess}}, \tag{4.1}$$

where t_i means the time score of the algorithm in this one test. However, we can do more tests - we have a rich dataset.

The total time score can be determined as the median of an array of sorted results, where each result is calculated on a unique scanned document (for the whole dataset):

$$t = Me(T), \tag{4.2}$$

where T is a set of all test results (time scores) done on the whole dataset and $Me(T)$ is a median of T.

We can assume that the time score in this case will be relatively independent of the architecture of CPU.

In the same way, we can calculate the value of total error rate:

$$e_i = \frac{e_{alg}}{e_{tess}}. \tag{4.3}$$

The total error score can be determined as the median of an array of sorted results:

$$e = Me(E), \tag{4.4}$$

where E is a set of all test results (error rates) done on the whole dataset and $Me(E)$ is a median of E.

To evaluate memory requirements we must take maximum peak of the allocated memory of all tests and compare it to the reference value given by Tesseract OCR engine:

$$m = \frac{max(M)}{max(M_{tess})}, \tag{4.5}$$

where M is a set of memory peaks of all tests for the whole dataset and M_{tess} is memory required by the Tesseract OCR engine.

4.2 Algorithm performance presentation

After performing the tests, for each algorithm we obtain a triple (e, m, t). For general purposes, we may need only one value, which must be calculated from this triple. Basically, it is an alternative to some benchmark value in the CPU world (Whetstone, Dhrystone, MIPS, FLOPS and other metrics). This benchmark value can be calculated as:

$$score = w_e e + w_m m + w_t t, \tag{4.6}$$

where w_e, w_m and w_t are constants. After some tests these values were chosen: $w_e = 0.8$, $w_t = 0.15$, $w_m = 0.05$.

4.3 Text similarity

To calculate our total error score we have to determine the way of document comparing.

Levenshtein distance is a metric that measures the difference between two sequences of characters. It is defined as the minimum number of one-character operations (insertion, deletion, replacement) needed to transform one sequence of characters into another.

Levenshtein's distances between words or text fields have the following disadvantages:

- rearranging words or parts of words gives relatively large distances;
- distances between completely different short words are small, while distances between very similar long words are considerable.

The calculation of the Levenshtein distance is based on the observation that if we reserve a matrix to store the Levenshtein distances between all prefixes of the first row and all prefixes of the second row, we can compute the values in the matrix using dynamic programming, and thus find the distance between two complete rows as the last computed value.

There is a sequence of actions required to get from the one string to another in the shortest possible way. Usually actions are denoted as: **D** (delete), **I** (insert), **R** (replace), **M** (match).

For example, for lines "CONNECT" and "CONEHEAD" we can build the following table of transformations which illustrated in the figure 4.1.

M	**M**	**M**	**R**	**I**	**M**	**R**	**R**
C	**O**	**N**	**N**		**E**	**C**	**T**
C	O	N	**E**	**H**	E	**A**	**D**

Table 4.1: Levenstein transformations table

Let S_1 and S_2 be two strings with length M and N respectively, then Levenstein distance $d(S_1, S_2)$ can be calculated by the following recurrence formula:

$$d(S_1, S_2) = D(M, N), \tag{4.7}$$

where $D(i,j)$ is defined as follows:

$$D(i,j) = \begin{cases} 0, & i=0, j=0 \\ i, & j=0, i>0 \\ j, & i=0, j>0 \\ min \begin{cases} D(i,j-1)+1 \\ D(i-1,j)+1 \\ D(i-1,j-1)+m(S_1[i],S_2[j]), \end{cases} & j>0, i>0 \end{cases} \tag{4.8}$$

where $m(a,b)$ is 0 if $a=b$ and 1 otherwise.

Each step by i represents deletion (**D**) from the first line, by j represents insertion (**I**) into the first line, and the step of both indexes represents character substitution (**R**) or no change (**M**).

Levenshtein similarity can be calculated by formula:

$$sym = \frac{d(S_1,S_2)}{max(M,N)}, \tag{4.9}$$

where S is a compared document consisting of consecutive lines, $max(M,N)$ the length of the longest word.

An example of the resulting matrix can be seen in table 4.2.

		S	**a**	**t**	**u**	**r**	**d**	**a**	**y**
	0	1	2	3	4	5	6	7	8
S	1	**0**	**1**	**2**	3	4	5	6	7
u	2	1	1	2	**2**	3	4	5	6
n	3	2	2	2	3	**3**	4	5	6
d	4	3	3	3	3	4	**3**	4	5
a	5	4	3	4	4	4	4	**3**	4
y	6	5	4	4	5	5	5	4	**3**

Table 4.2: An example of how the Levenshtein algorithm works.

So our error rate is calculated by:

$$e = 1 - sym. \tag{4.10}$$

4.4 Character Error Rate

Character Error Rate (CER) is a common performance metric for automatic speech recognition systems. However, it is also suitable for our text recognition

goals. The character error rate [30] can be calculated as follows:

$$CER = \frac{(\mathbf{R} + \mathbf{D} + \mathbf{I})}{\mathbf{N}} = \frac{(\mathbf{R} + \mathbf{D} + \mathbf{I})}{(\mathbf{R} + \mathbf{D} + \mathbf{M})}, \tag{4.11}$$

where $\mathbf{N}$ is the number of characters in the reference ($\mathbf{N} = \mathbf{R} + \mathbf{D} + \mathbf{M}$) - other values are defined before.

Chapter 5

Data extraction

For high-quality image processing, a careful approach to preprocessing is needed, as it has a big impact on the final result. Before classifying the text, it is first necessary to separate from the background and create a dataset, which become the basis for training and validating the OCR algorithms.

Earlier in this paper we discussed various methods of image segmentation, which are widely used to solve different problems. In order to understand which method is suitable for historical archives processing, I decided to test the relevance of their application to our data.

5.1 U-net processing

The network is trained on a small number of images and outperformes the previous best method (a convolutional network with a sliding window) in the ISBI competition for segmentation of neural structures in electron microscopic stacks. U-Net won first place in the ISBI 2015 in cell-tracking competition by a landslide. In addition, this network is fast. Segmenting a 512×512 image takes less than a second on a modern GPU. It is also important that the U-net is characterized by using a small amount of data to achieve good results.

5.1.1 Architecture

U-Net is considered one of the standard CNN architectures for image segmentation tasks, when it is necessary not only to define an entire image class, but also to segment its areas by class, i.e. to create a mask that will divide the image into several classes. Unfortunately this architecture cannot cope with the classification of text, so we will do with single-class classification (separating the background from the background).The architecture consists of a pooling path for context extracting and a symmetric expanding path that allows precise localization. The table C.1 shows the detailed architecture used in the computation.

5.1.2 Training

Before training our network we need to annotate data. I did it with Thresholding method, selecting the most appropriate parameters manually. Since training takes a very long time, I decided to bring the data to a size of 512×512 pixels. An example is given in figure 5.1.

Figure 5.1: Training data for U-net.

The network is trained by stochastic gradient descent based on the input images and their corresponding segmentation maps. Due to convolutions, the output image is smaller than the input by a constant boundary width. The calculations also use pixel-wise soft-max over the final feature map combined with the cross entropy loss function.

The partition boundary is calculated using morphological operations. Then the weighting map is calculated by:

$$w(x) = w_c(x) + w_0 * exp(-\frac{(d_1(x) + d_2(x))^2}{2\sigma^2}), \tag{5.1}$$

where w_c is the weight map to balance the class frequencies, d_1 means the distance to the border of the nearest foreground part, d_2 the distance to the second nearest part, w_0 and σ are set manually, by default they are 10 and 5 pixels respectively [11].

In my experiments, I used auxiliary methods from the keras_unet external python library. The data was split to train and validation parts. The results of training on 10 epochs are presented in figures 5.2 and 5.3.

5.1.3 Prediction

After long enough training on the training data, U-net showed high performance in time of mask prediction. The results are shown in figure 5.4.

5.2 Segmentation methods comparison

After training U-net I decided to try Thresholding and Canny edge detection methods for extracting useful features. OpenCV library provides the function-

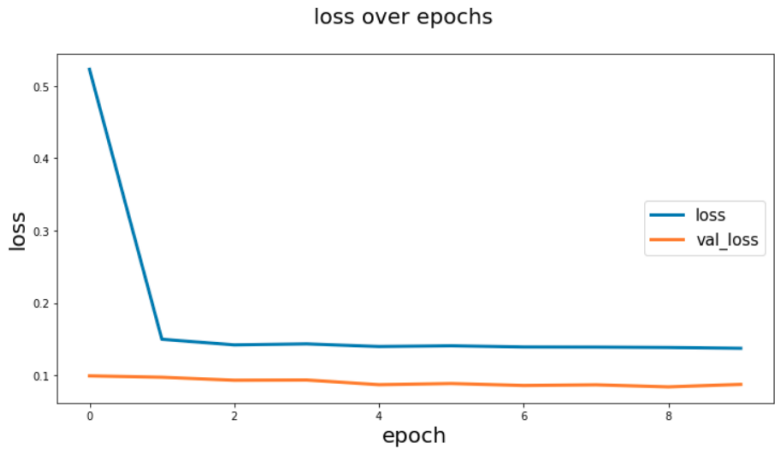


Figure 5.2: U-net training overview with binary crossentropy loss function.

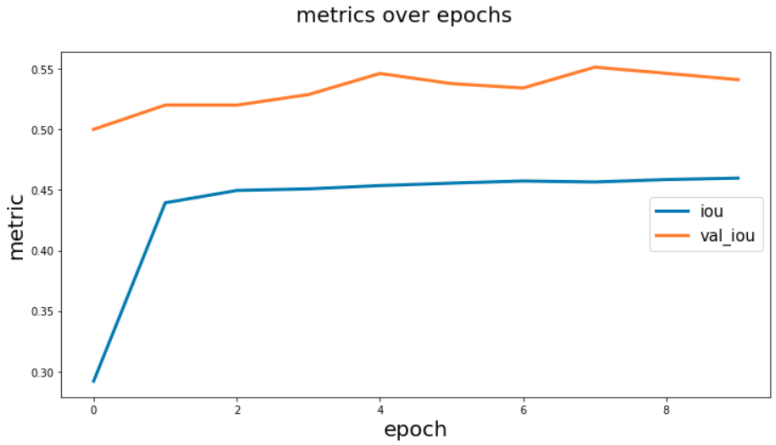


Figure 5.3: U-net training overview with Intersection over Union metric.

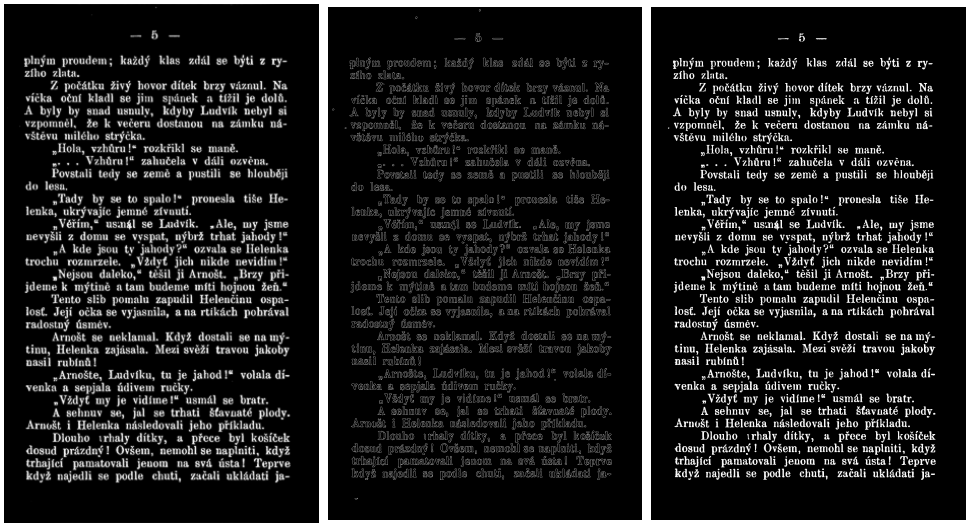
ality to work with these algorithms. I used 60 and 250 (pixel brightness) as threshold values for both algorithms.

It's hard to say right away which one will work best with our dataset. So I decided to test the segmentation quality by evaluating the recognition with a strong OCR algorithm (Tesseract with LSTM integration). I sent each of the masks as input to this engine. The digitized images were converted to text and then I compared each of them with the correct text.

I decided to measure the recognition error with the Levenstein similarity described earlier in this thesis. The results of the calculations are in table 5.1.

Thresholding showed the best result in segmenting the text, so further this method will be used.

Figure 5.4: Examples of masks predicted by U-net.



a) U-net b) Canny c) Thresholding

Figure 5.5: Segmentation masks by different algorithms.

	Original	**Thresholding**	**Canny**	**U-net**
error	0.02005	0.01259	0.2183	0.8537

Table 5.1: Evaluation of different segmentation methods. Error based on Levenstein similarity

5.3 Forming dataset

I used the Kramerius (Digital Library of the National Library of the CR) as a data source. The digital library includes several million digitized pages of monographs and periodicals from the collections of the National Library of the Czech Republic, which were digitized as part of the National Digital Library project. A 19th century Czech book named "Větévky z útlého kmene povídky milé mládeži" [31] was used as czech language dataset base.

I downloaded djvu images from Kramerius website, converted them to png images files for easier processing.

After choosing segmentation method, the images should be processed to improve the recognition result. I used a bounding box to separate parts of the text from each other (figure 5.6). The bounding box is an imaginary minimal rectangle that serves as a reference point for object detection.

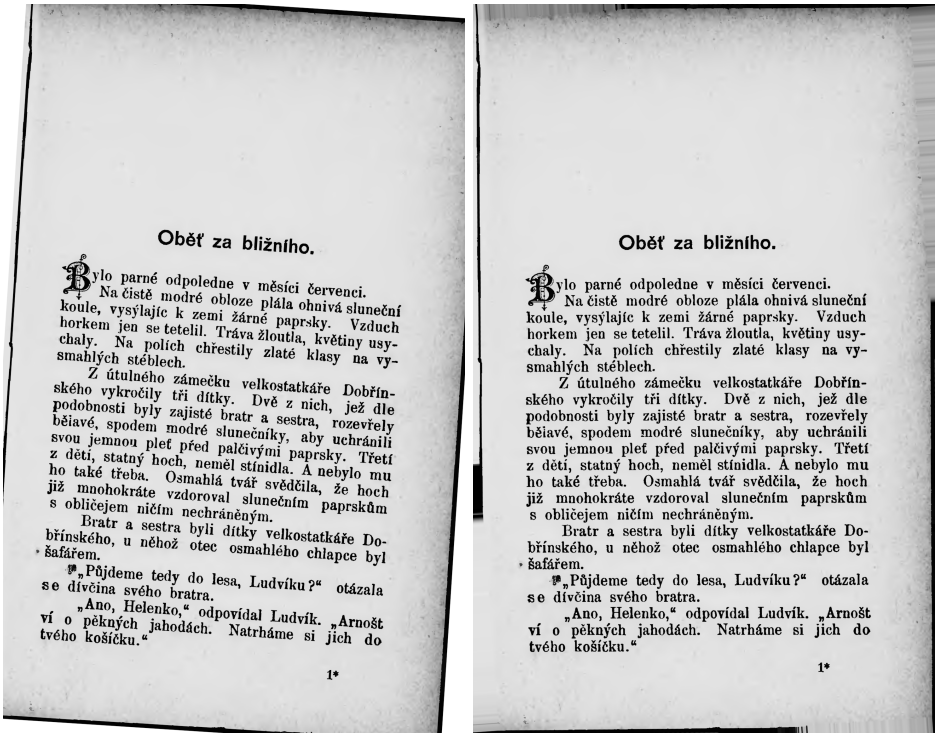
Figure 5.6: Bounding boxes example.

Throughout the experiments I also used grayscaling, deskewing (figure 5.7), different methods of noise reduction (combining filters as well as morphological operations) to achieve the best possible result.

5.3.1 Splitting page text into lines

In order to select lines on the page I used morphological operations, such as *dilation* and *opening*. Applying dilate to merge text into meaningful lines/paragraphs. I used larger kernel on x axis $(120, 1)$ to merge characters into single line, cancelling out any spaces but smaller kernel on y axis $(1, 3)$ to separate between different lines.

Then I wrote a function that aligns contours in the order in which a person would read them. It converts the information about the contours into a rank and uses that rank to sort the contours. The rank changes a lot when two consecutive contours lie vertically, but changes only slightly if the contours are horizontal. The contours are grouped from top to bottom first, and the lowest



Oběť za bližního.

Bylo parné odpoledne v měsíci červenci. Na čistě modré obloze plála ohnivá sluneční koule, vysýlajíc k zemi žárné paprsky. Vzduch horkem jen se tetelil. Tráva žloutla, květiny usychaly. Na polích chřestily zlaté klasy na vysmahlých stéblech.

Z útulného zámečku velkostatkáře Dobřínského vykročily tři dítky. Dvě z nich, jež dle podobnosti byly zajisté bratr a sestra, rozevřely běiavé, spodem modré slunečníky, aby uchránili svou jemnou pleť před palčivými paprsky. Třetí z dětí, statný hoch, neměl stínidla. A nebylo mu ho také třeba. Osmahlá tvář svědčila, že hoch již mnohokráte vzdoroval slunečním paprskům s obličejem ničím nechráněným.

Bratr a sestra byli dítky velkostatkáře Dobřínského, u něhož otec osmahlého chlapce byl šafářem.

„Půjdeme tedy do lesa, Ludvíku?“ otázala se dívčina svého bratra.

„Ano, Helenko,“ odpovídal Ludvík. „Arnošt ví o pěkných jahodách. Natrháme si jich do tvého košíčku.“

1*

a) Skewed image b) Fixed rotation

Figure 5.7: Deskewing image.

variant value among horizontally stacked contours is used in case of a collision. Example may be seen at figure 5.8.

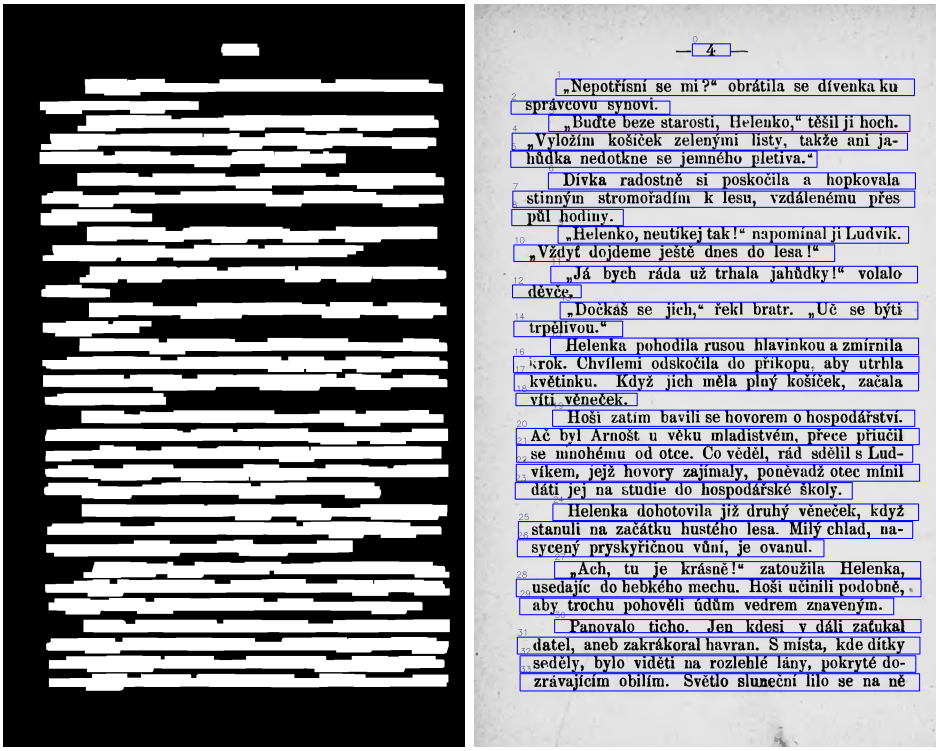
After that, I divided the pre-prepared files with the correct text into separated text lines and matched them with pictures. It formed ground truth data for our dataset.

Therefore, we can process the entire data archive, resulting in a massive dataset of characters. For the tested algorithms division into strings was enough, but for future modifications of this project it will be useful to test other algorithms.

By running the algorithm on scanned lines, the scanned photos of which contain inaccuracies and artifacts, I got the following result (figure 5.9).

5.3.2 Book format

For further extensions of the project, I also added the ability to add books. This is done by calling a special method of the class created as part of the environment for testing. Based on the books from Kramerius, I identified the main important data for further page processing and testing of various algorithms.



a) segmented lines b) numbered lines

Figure 5.8: Splitting page into lines.

Figure 5.9: Letters for dataset.

After the function is called, a data folder and a JSON file (figure 5.10 are created in the testing environment folder.

```
{
    "main_title": "Vetevky z utleho kmene",
    "sub_title": "povidky mile mladezi",
    "creator_surname": "Sastny",
    "creator_name": "Alfons, Bohumil",
    "lang": "ces"
}
```

Figure 5.10: Book as JSON example.

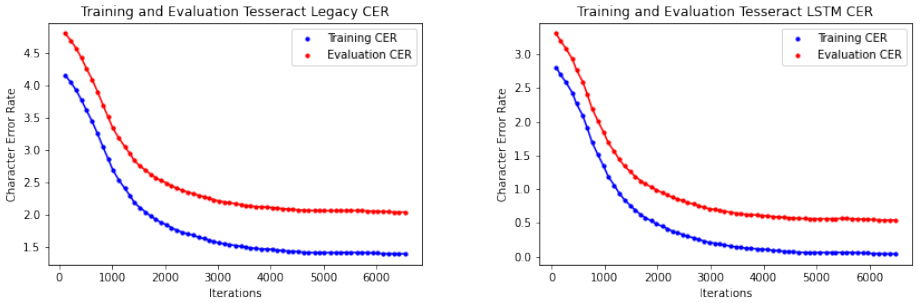
Chapter 6

Results

In this chapter the results of the algorithms will be evaluated using a testing environment. The classification accuracy during training of some algorithms will also be evaluated.

6.1 Training state-of-the art engine

The Tesseract has been trained on a sufficiently large amount of data and shows impressive results in the text recognition task. In spite of this, the accuracy of the algorithm can be improved even more by the transfer learning method described in the theoretical part of the paper. I took the standard Czech language model and trained it on data from our dataset using the official Tesseract software called *tesstrain*. As training data I used the previously extracted strings, as well as their corresponding labels, i.e. ground truth text files. Figure 6.1 show the loss functions during training of different modes of the Tesseract engine.



a) Tesseract Legacy b) Tesseract LSTM

Figure 6.1: Tesseract training results.

The value of this function should decrease during the learning process. The

curve of the loss function value during the training phases ideally should be without significant jumps or breaks. These are exactly the results I was able to achieve while training both algorithms. As expected, LSTM showed better results.

Looking at the loss functions of both models, we can also see that the functions converge approximately after the first third of the training process. This indicates that after this time, the algorithm learned, so to speak, "in vain" and therefore its training could have been terminated earlier.

6.2 Testing algorithms

After training algorithms, they can be used for text prediction then. Implemented testing environment allows us to add new books without much difficulty. So I decided to add books in German and English for better clarity in comparing the algorithms.

I chose Tesseract LSTM as the benchmark algorithm because it showed good results during the training. The other algorithms will be relative to it on the parameters of recognition success, speed, and memory, as described earlier in the thesis.

After running Tesseract LSTM, a JSON file named 'tesseract.json' appears in the 'metrics_data' book folder. An example of the content of this file is shown in the figure 6.2.


```
{
    "t_tess_list":
        [2.0649, 1.4370, 1.4773,
        1.4746, 1.5845, 1.6012,
        1.2066, 1.4560, 1.2688],
    "e_tess_list":
        [0.0239, 0.0193, 0.02487,
        0.01580, 0.01905, 0.0203,
        0.01966, 0.0058, 0.01832],
    "m_tess": 161.6796
}
```


Figure 6.2: Tesseract metrics as JSON example.

If user choose to test any of the algorithms, the environment implies the generation of a file that includes its effectiveness data. I also added information fields for clarity, such as the name of the book, the name of the algorithm, the language of recognition. Content of such file demonstrated in figure 6.3.

```
{
    "book": "vetevky_z_utleho_kmene",
    "algorithm": "GOCR",
    "lang": "ces",
    "t": 0.7458239817573651,
    "e": 26.46705309342672,
    "m": 1.196762503020053,
    "score": 21.345354197155984
}
```


Figure 6.3: Algorithm evaluation as JSON example.

6.2.1 Tesseract LSTM

The most effective framework has proved to be Tesseract LSTM. Its advanced deep learning implementation has allowed it to surpass other OCR algorithms. After training, its accuracy became about 98%. In figure 6.1 is Tesseract LSTM testing score.

Book	**Metrics**	**Values**								
cze	e	2.06	1.43	1.47	1.47	1.58	1.60	1.20	1.45	1.26
	t	1.82	2.67	1.61	2.21	2.81	3.06	2.04	2.57	2.92
	m	161.67								
eng	e	1.82	2.67	1.61	2.21	2.81	3.06	2.04	2.57	2.92
	t	0.04	0.02	0.07	0.04	0.02	0.02	0.04	0.03	0.02
	m	202.18								
deu	e	1.11	1.77	1.32	2.08	1.66	1.60	1.41	1.23	1.08
	t	0.34	0.44	0.36	0.43	0.44	0.40	0.44	0.36	0.42
	m	170.67								

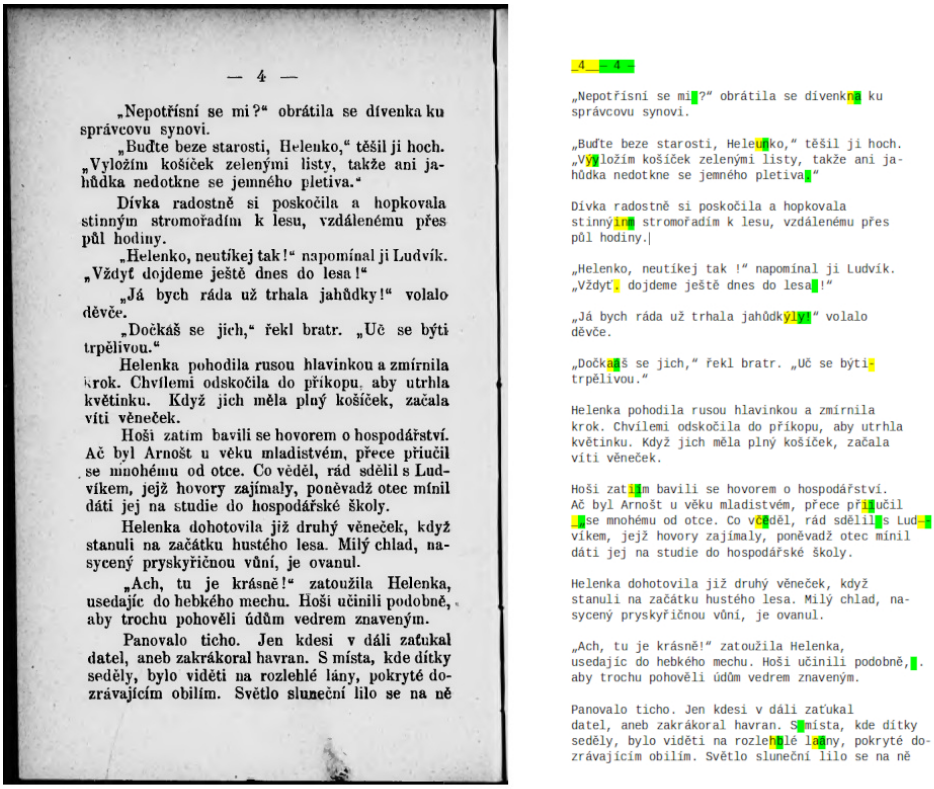
Table 6.1: Tesseract evaluating variables.

All of the above metrics were described in the chapter 4. Error metric e based on Levenstein similarity.

6.2.2 Tesseract Legacy

As can be seen in figure 6.4, the Tesseract Legacy scored about 1.5, which means that it is 1.5 times worse than the benchmark Tesseract LSTM.

An interesting detail is that the Legacy mode engine required the same amount of memory to process the text.



— 4 —

„Nepotřísní se mi?“ obrátila se dívenka ku správcovu synovi.

„Buďte beze starosti, Helenko,“ těšil ji hoch. „Vyložím košíček zelenými listy, takže ani jahůdka nedotkne se jemného pletiva.“

Dívka radostně si poskočila a hopkovala stinným stromořadím k lesu, vzdálenému přes půl hodiny.

„Helenko, neutíkej tak!“ napomínal ji Ludvík. „Vždyť dojdeme ještě dnes do lesa!“

„Já bych ráda už trhala jahůdky!“ volalo děvče.

„Dočkáš se jich,“ řekl bratr. „Uč se býti trpělivou.“

Helenka pohodila rusou hlavinkou a zmírnila krok. Chvilemi odskočila do příkopu, aby utrhla květinku. Když jich měla plný košíček, začala víti věneček.

Hoši zatím bavili se hovorem o hospodářství. Ač byl Arnošt u věku mladistvém, přece přiučil se mnohému od otce. Co věděl, rád sdělil s Ludvíkem, jejž hovory zajímaly, poněvadž otec mínil dáti jej na studie do hospodářské školy.

Helenka dohotovila již druhý věneček, když stanuli na začátku hustého lesa. Milý chlad, nasycený pryskyřičnou vůní, je ovanul.

„Ach, tu je krásně!“ zatoužila Helenka, usedajíc do hebkého mechu. Hoši učinili podobně, aby trochu pohověli údům vedrem znaveným.

Panovalo ticho. Jen kdesi v dáli zaťukal datel, aneb zakrákoral havran. S místa, kde dítky seděly, bylo viděti na rozlehlé lány, pokryté dozrávajícím obilím. Světlo sluneční lilo se na ně

_4_ — 4 —

„Nepotřísní se mi?“ obrátila se dívenka ku správcovu synovi.

„Buďte beze starosti, Heleuko,“ těšil ji hoch. „Vyložím košíček zelenými listy, takže ani jahůdka nedotkne se jemného pletiva.“

Dívka radostně si poskočila a hopkovala stinnýirá stromořadím k lesu, vzdálenému přes půl hodiny.

„Helenko, neutíkej tak !“ napomínal ji Ludvík. „Vždyt. dojdeme ještě dnes do lesa!“

„Já bych ráda už trhala jahůdkylyt“ volalo děvče.

„Dočkáš se jich,“ řekl bratr. „Uč se býti- trpělivou.“

Helenka pohodila rusou hlavinkou a zmírnila krok. Chvilemi odskočila do příkopu, aby utrhla květinku. Když jich měla plný košíček, začala víti věneček.

Hoši zatím bavili se hovorem o hospodářství. Ač byl Arnošt u věku mladistvém, přece přiučil se mnohému od otce. Co vědě̈l, rád sdělil s Ludvíkem, jejž hovory zajímaly, poněvadž otec mínil dáti jej na studie do hospodářské školy.

Helenka dohotovila již druhý věneček, když stanuli na začátku hustého lesa. Milý chlad, nasycený pryskyřičnou vůní, je ovanul.

„Ach, tu je krásně!“ zatoužila Helenka, usedajíc do hebkého mechu. Hoši učinili podobně, . aby trochu pohověli údům vedrem znaveným.

Panovalo ticho. Jen kdesi v dáli zaťukal datel, aneb zakrákoral havran. S místa, kde dítky seděly, bylo viděti na rozlehlé lány, pokryté dozrávajícím obilím. Světlo sluneční lilo se na ně

Figure 6.4: Tesseract processing result. Green is Tesseract LSTM result, yellow is Tesseract Legacy, uncolored characters both algorithms have the same.

Book	t	e	m	score
Větévky z útlého kmene (cze)	2.88	1.32	1.0	1.54
The victoria falls (eng)	1.67	1.42	1.0	1.44
Abhandlung über (deu)	3.16	1.25	1.0	1.52

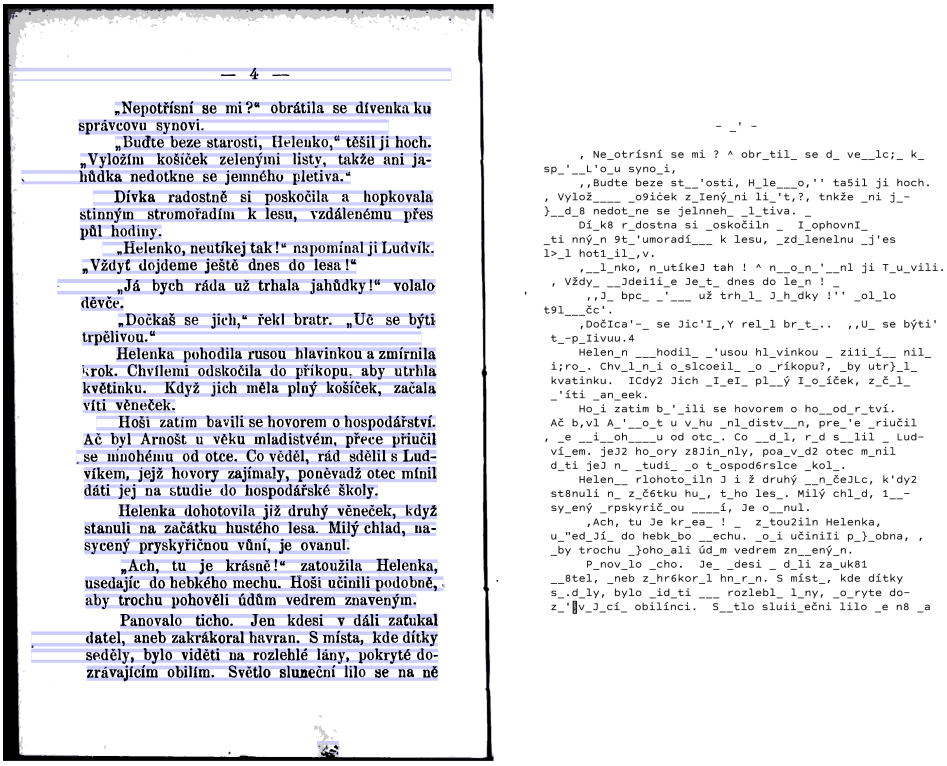
Table 6.2: Tesseract Legacy testing score

6.2.3 GOCR

Algorithm in process and result below (see fig. 6.5).

From the table of results in table 6.3 can be seen that it is 20–30 times worth then state-of-the-art algorithm. It can be concluded that it is bad enough for such noisy images.

Despite the fact that text processing is faster and sometimes less memory is spent, the recognition error is too high for this algorithm to be considered effective nowadays.



— 4 —

„Nepotřísní se mi?“ obrátila se dívenka ku správcovu synovi.

„Buďte beze starosti, Helenko,“ těšil ji hoch. „Vyložím košíček zelenými listy, takže ani jahůdka nedotkne se jemného pletiva.“

Dívka radostně si poskočila a hopkovala stinným stromořadím k lesu, vzdálenému přes půl hodiny.

„Helenko, neutíkej tak!“ napomínal ji Ludvík. „Vždyť dojdeme ještě dnes do lesa!“

„Já bych ráda už trhala jahůdky!“ volalo děvče.

„Dočkaš se jich,“ řekl bratr. „Uč se býti trpělivou.“

Helenka pohodila rusou hlavinkou a zmírnila krok. Chvílemi odskočila do přikopu, aby utrhla květinku. Když jich měla pluý košíček, začala víti věneček.

Hoši zatím bavili se hovorem o hospodářství. Ač byl Arnošt u věku mladistvém, přece přiučil se mnohému od otce. Co věděl, rád sdělil s Ludvíkem, jejž hovory zajímaly, poněvadž otec mínil dáti jej na studie do hospodářské školy.

Helenka dohotovila již druhý věneček, když stanuli na začátku hustého lesa. Milý chlad, nasycený pryskyřičnou vůní, je ovanul.

„Ach, tu je krásně!“ zatoužila Helenka, usedajíc do hebkého mechu. Hoši učinili podobně, aby trochu pohověli údům vedrem znaveným.

Panovalo ticho. Jen kdesi v dáli zaťukal datel, aneb zakrákoral havran. S místa, kde dítky seděly, bylo viděti na rozlehlé lány, pokryté dozrávajícím obilím. Světlo sluneční lilo se na ně

- _' -

, Ne_otrísní se mi ? ^ obr_til_ se d_ ve__lc;_ k_
sp_'__L'o_u syno_i,
,,Budte beze st__'osti, H_le___o,'' tašil ji hoch.
, Vylož____ _o9iček z_Iený_ni li_'t,?, tnkže _ni j_-
}__d_8 nedot_ne se jelnneh_ _l_tiva. _
Dí_k8 r_dostna si _oskočiln _ I_ophovnI_
_ti nný_n 9t_'umoradí___ k lesu, _zd_lenelnu _j'es
l>_l hot1_il_,v.
,__l_nko, n_utíkeJ tah ! ^ n__o_n_'__nl ji T_u_vili.
, Vždy_ __Jdei1i_e Je_t_ dnes do le_n ! _
,,J_ bpc_ _'___ už trh_l_ J_h_dky !'' _ol_lo
t9l___čc'.
,Dočica'-_ se Jic'I_,Y rel_l br_t_.. ,,U_ se býti'
t_-p_Iivuu.4
Helen_n ___hodil_ _'usou hl_vinkou _ ziii_í__ nil_
i;ro_. Chv_l_n_i o_slcoeil_ _o _ríkopu?, _by utr}_l_
kvatinku. ICdy2 Jich _I_eI_ pl__ý I_o_íček, z_č_l_
_'íti _an_eek.
Ho_i zatim b_'_ili se hovorem o ho__od_r_tví.
Ač b,vl A_'__o_t u v_hu _nl_distv__n, pre_'e _riučil
, _e __i__oh___u od otc_. Co __d_l, r_d s__lil _ lud-
ví_em. jeJ2 ho_ory z8Jin_nly, poa_v_d2 otec m_nil
d_ti jeJ n_ _tudi_ _o t_ospod6rslce _kol_.
Helen__ rlohoto_iln J i ž druhý __n_čeJLc, k'dy2
st8nuli n_ z_čétku hu_, t_ho les_. Milý chl_d, 1__-
sy_ený _rpskyrič_ou ____í, Je o__nul.
,Ach, tu Je kr_ea_ ! _ z_tou2iln Helenka,
u_"ed_Jí_ do hebk_bo __echu. _o_i učiniIi p_}_obna, ,
_by trochu _}oho_ali úd_m vedrem zn__ený_n.
P_nov_lo _cho. Je_ _desi _ d_li za_uk81
__8tel, _neb z_hr6kor_l hn_r_n. S míst_, kde dítky
s_.d_ly, bylo _id_ti ___ rozlebl_ l_ny, _o_ryte do-
z_'v_J_cí_ obilínci. S__tlo sluii_ební lilo _e n8 _a

Figure 6.5: GOCR Processing result.

Book	t	e	m	score
Větévky z útlého kmene (cze)	0.74	26.46	1.19	21.34
The victoria falls (eng)	2.52	28.25	0.95	23.02
Abhandlung über (deu)	0.68	37.19	1.0	29.90

Table 6.3: GOCR testing score

Conclusion

The first part of this thesis focused on an overview of image processing methods. As it was found during the study preprocessing plays a major role in text recognition process.

Part of the research was devoted to soft computing, namely artificial intelligence and machine learning methods. The biggest focus of this theoretical part was on neural networks. Some important deep learning architectures that are used in modern text recognition systems were explored, as well as the analysis of the learning process, some machine learning metrics.

The purpose of this work was to evaluate existing text recognition algorithms.

For example, I investigated both classical text recognition approaches and approaches using soft computing, such as LSTM neural networks. A review of commercial and non-commercial programs was also conducted.

I used the Kramerius Digital Library as a data source to form dataset. To train the OCR model in the most efficient way possible I have also tested some segmentation algorithms and found the most appropriate one under our conditions to apply it for extracting useful data from parsed dataset.

A way to test different algorithms was realized. In the course of the research it was found that in today's realities of the text recognition market there is a high competitiveness. Therefore, it is important to know how the algorithms differ and to what extent they outperform each other.

Metrics were derived that take one of the best opensource engines Tesseract LSTM for text recognition as a benchmark.

The development rudiments of this project in the future were also laid out. New algorithms for text recognition are not difficult to add to testing environment. A dataset of 10,000 letters was also formed to test other algorithms in the future. The possibility of adding one's own data has been implemented, which simplifies the creation of the dataset in the future. Potential user can train OCR algorithms on his own data, which, of course, can be non-alphanumeric characters, Chinese characters, Cyrillic characters.

Bibliography

[1] Dumoulin, V.; Visin, F. A guide to convolution arithmetic for deep learning. *arXiv*, 2016, ISSN 1603-07285.

[2] Artyomov, A. Convolutional neural networks [online]. sep 2016, [cit. 20.5.2021]. Available from: `http://www.machinelearning.ru/wiki/images/1/1b/DL16_lecture_3.pdf`

[3] Boiangiu, C. A.; Ioanitescu, R.; et al. VOTING-BASED OCR SYSTEM. *Journal of Information Systems and Operations Management*, volume 10, 12 2016: pp. 470–486, ISSN 18434711.

[4] Smith, R. An overview of the Tesseract OCR engine. In *Ninth international conference on document analysis and recognition (ICDAR 2007)*, volume 2, IEEE, 2007, ISSN 1520-5363, pp. 629–633.

[5] Smith, R. Tesseract documentation [online]. 2016, [cit. 5.4.2021]. Available from: `https://github.com/tesseract-ocr/docs/blob/master/das_tutorial2016/6ModernizationEfforts.pdf`

[6] Schulenburg, J. GOCR [online]. [cit. 25.4.2021]. Available from: `http://jocr.sourceforge.net/`

[7] reference, O. Garbage in garbage out [online]. Available from: `https://www.oxfordreference.com/view/10.1093/oi/authority.20110803095842747`

[8] McAndrew, A. *A computational introduction to digital image processing*. CRC Press, 2015, ISBN 97-804-291-618-03.

[9] OpenCV developers team. *OpenCV 2.4.13.7 documentation [online]*. 2011-2014, [cit. 6.4.2021]. Available from: `https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html`

[10] Thipkham, P. Image Processing Class 4 — Filters [online]. dec 2018, [cit. 06.04.2021]. Available from: https://towardsdatascience.com/image-processing-class-egbe443-4-filters-aa1037676130

[11] Ronneberger, O.; Fischer, P.; et al. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, Springer, 2015, ISBN 97-833-192-457-37, pp. 234–241.

[12] Long, J.; Shelhamer, E.; et al. Fully convolutional networks forsemantic segmentation. *CoRR*, 2014, ISSN 1063-6919.

[13] Canny, J. A Computational Approach to Edge Detection. *Encyclopedia of Statistical Sciences*, volume 6, 11 1986: pp. 679–697, ISSN 0162-8828.

[14] Gupta, M. M. *Soft computing and intelligent systems: theory and applications*. Elsevier, 1999, ISBN 97-801-264-649-00, 77-—84 pp.

[15] Ulyanov, S. V. Intelligent Robust Control: Soft Computing Technologies. *PronetLabs*, 2011: p. 406, ISSN 978-5-8481-0075-4.

[16] Yegnanarayana, B. *Artificial neural networks*. PHI Learning Pvt. Ltd., 2009, ISBN 81-203-1253-8.

[17] Rutkowska, D.; Pilinski, M.; et al. Neural networks, genetic algorithms and fuzzy systems. *Sieci neuronowe, algorytmy genetyczne i systemy rozmyte*, 2013, ISSN 5-93517-103-1.

[18] Smítka, J.; Borkovcová, M. *Automatic recognition of manuscripts and other historical materials. Reviewed proceedings of the International Masaryk Conference for PhD. Students and Young Researchers*. Hradec Králové, Czech Republic: MAGNANIMITAS, vol. 10, 2019, ISBN 97-880-879-523-13.

[19] Olah, C. Understanding lstm networks [online]. aug 2015, [cit. 10.06.2021]. Available from: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[20] Werbos, P. J. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, volume 78, no. 10, 1990: pp. 1550–1560, ISSN 0018-9219.

[21] Hochreiter, S.; Schmidhuber, J. Long Short-term Memory. *Neural computation*, volume 9, 12 1997: pp. 1735–80, ISSN 1735–1780.

[22] Xiao, F.; Honma, Y.; et al. A simple algebraic interface capturing scheme using hyperbolic tangent function. *International Journal for Numerical Methods in Fluids*, volume 48, no. 9, 2005: pp. 1023–1040, ISSN 0021-9991.

[23] West, J.; Ventura, D.; et al. Spring research presentation: A theoretical foundation for inductive transfer. *Brigham Young University, College of Physical and Mathematical Sciences*, volume 1, no. 08, 2007, ISSN 1059-7123.

[24] Duin, R. P.; Haringa, H.; et al. Fast percentile filtering. *Pattern recognition letters*, volume 4, no. 4, 1986: pp. 269–272, ISSN 0167-8655.

[25] Schneider, P. J. An algorithm for automatically fitting digitized curves. *Graphics gems*, volume 1, 1990: pp. 612–626, ISSN 978-0-08-050753-8.

[26] Nagy, G.; Nartker, T. A.; et al. *Optical character recognition: An illustrated guide to the frontier*, volume 3967. Kluwer Academic Publishers, 1999, ISBN 978-1-4615-5021-1, 58–69 pp.

[27] Smith, R. W. *The extraction and recognition of text from multimedia document images*. Dissertation thesis, University of Bristol, 1987.

[28] Hart, P. E.; Nilsson, N. J.; et al. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, volume 4, no. 2, 1968: pp. 100–107, ISSN 2168-2887.

[29] TopSoft Inc. and TopOCR. Top Soft's user's guide [online]. *Softtop*, 2009, [cit. 20.06.2021]. Available from: `http://www.topocr.com`

[30] Morris, A. C.; Maier, V.; et al. From WER and RIL to MER and WIL: improved evaluation measures for connected speech recognition. *Eighth International Conference on Spoken Language Processing*, 2004: pp. 1–3.

[31] ŠŤASTNÝ, A. B. *Větévky z útlého kmene: povídky milé mládeži*, volume 3. V Praze: M. Knapp, 1892, 3–11 pp.

Appendix A

Acronyms

OCR Optical Character Recognition

CNN Convolutional Neural Network

RNN Recurrent Neural Network

LSTM Long Short-Term Memory

ReLU Rectified Linear Unit

Appendix B

Contents of enclosed CD

- `readme.md` ..... the file with CD contents description
- `books` ..... the directory with books
- `pictures` ..... the directory of source codes
- `training` ..... the directory of model training files
  - `train` ..... train data
  - `evaluate` ..... evaluation data
- `characters_dataset` ..... the dataset of recognized character
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format
- `notebooks` ..... jupyter notebooks related to work

Appendix C

Appendix

Layer (type)	Output shape	Param #	Connected to
input_2	(512, 512, 1)	0	
conv2d_20	(512, 512, 64)	640	input_2
dropout_6	(512, 512, 64)	0	conv2d_20
conv2d_21	(512, 512, 64)	36928	dropout_6
max_pooling2d_5	(256, 256, 64)	0	conv2d_21
conv2d_22	(256, 256, 128)	73856	max_pooling2d_5
dropout_7	(256, 256, 128)	0	conv2d_22
conv2d_23	(256, 256, 128)	147584	dropout_7
max_pooling2d_6	(128, 128, 128)	0	conv2d_23
conv2d_24	(128, 128, 256)	295168	max_pooling2d_6
dropout_8	(128, 128, 256)	0	conv2d_24
conv2d_25	(128, 128, 256)	590080	dropout_8
max_pooling2d_7	(64, 64, 256)	0	conv2d_25
conv2d_26	(64, 64, 512)	1180160	max_pooling2d_7
dropout_9	(64, 64, 512)	0	conv2d_26
conv2d_27	(64, 64, 512)	2359808	dropout_9
max_pooling2d_8	(32, 32, 512)	0	conv2d_27
conv2d_28	(32, 32, 1024)	4719616	max_pooling2d_8
dropout_10	(32, 32, 1024)	0	conv2d_28
conv2d_29	(32, 32, 1024)	9438208	dropout_10
conv2d_transpose_5	(64, 64, 512)	2097664	conv2d_29
concatenate_5	(64, 64, 1024)	0	conv2d_transpose_5 conv2d_27
conv2d_30	(64, 64, 512)	4719104	concatenate_5
conv2d_31	(64, 64, 512)	2359808	conv2d_30
conv2d_transpose_6	(128, 128, 256)	524544	conv2d_31
concatenate_6	(128, 128, 512)	0	conv2d_transpose_6 conv2d_25
conv2d_32	(128, 128, 256)	1179904	concatenate_6
conv2d_33	(128, 128, 256)	590080	conv2d_32
conv2d_transpose_7	(128, 128, 128)	131200	conv2d_33
concatenate_7	(256, 256, 256)	0	conv2d_transpose_7 conv2d_23
conv2d_34	(256, 256, 128)	295040	concatenate_7
conv2d_35	(256, 256, 128)	147584	conv2d_34
conv2d_transpose_8	(512, 512, 64)	32832	conv2d_35
concatenate_8	(512, 512, 128)	0	conv2d_transpose_8 conv2d_21
conv2d_36	(512, 512, 64)	73792	concatenate_8
conv2d_37	(512, 512, 64)	36928	conv2d_36
conv2d_38	(512, 512, 1)	65	conv2d_37

Table C.1: U-net model architecture, Total params: 31 030 593.