České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská

Katedra softwarového inženýrství

Obor: Aplikace informatiky v přírodních vědách

Optické rozpoznávání znaků na naskenovaných historických plakátech pomocí nejmodernějších metod

Optical Character Recognition on Scanned Historical Posters Using the State-of-the-Art Methods

VÝZKUMNÝ ÚKOL

Vypracoval: Anna Gruberová

Vedoucí práce: Ing. Adam Novozámský, Ph.D.

Rok: 2022

České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská

Katedra softwarového inženýrství

Akademický rok 2021/2022

ZADÁNÍ VÝZKUMNÉHO ÚKOLU

Student:	**Bc. Anna Gruberová**
Studijní program:	**Aplikace informatiky v přírodních vědách**
Název práce česky:	**Optické rozpoznávání znaků na naskenovaných historických plakátech pomocí nejmodernějších metod**
Název práce anglicky:	**Optical Character Recognition on Scanned Historical Posters Using the State-of-the-Art Methods**

Pokyny pro vypracování:

1. Seznamte se s problematikou optického rozpoznávání znaků. Na základě rešerše vyberte několik metod, se kterými budete dále pracovat a vyhodnocovat úspěšnost jejich detekce.
2. Stáhněte několik volně dostupných datasetů, které jsou využívány v literatuře k porovnání jednotlivých metod na OCR. Dále vytvořte svůj vlastní dataset z obdržených dat.
3. Nastudujte techniky porovnání OCR výstupů s ground-truth.
4. U vybraných metod prostudujte jejich chování na jednotlivých datasetech při různém nastavení parametrů.
5. Navrhněte také možnosti filtrování výstupů jednotlivých metod za účelem snížení falešných detekcí.

Doporučená literatura:

[1] R. C. Gonzalez, R. E. Woods, Digital Image Processing (4th ed.). Pearson, 2018. ISBN 9353062985.
[2] GOODFELLOW, Ian, Yoshua BENGIO a Aaron COURVILLE. Deep learning. Cambridge, Massachusetts: The MIT Press, [2016]. ISBN 0262035618.
[3] SMITH, R. An Overview of the Tesseract OCR Engine. In: Ninth International Conference on Document Analysis and Recognition (ICDAR 2007) Vol 2 [online]. IEEE, 2007, 2007, s. 629-633. ISBN 0-7695-2822-8. ISSN 1520-5363. Dostupné z: doi:10.1109/ICDAR.2007.4376991
[4] CHEN, Xiaoxue, et al. Text Recognition in the Wild. ACM Computing Surveys [online]. 2021, 54(2), 1-35 [cit. 2021-10-2]. ISSN 0360-0300. Dostupné z: doi:10.1145/3440756

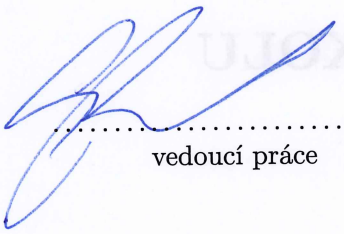
Jméno a pracoviště vedoucího práce:

Ing. Adam Novozámský, Ph.D.
Computer Vision Lab, Institute of Visual Computing & Human-Centered Technology,
TU Wien - Faculty of Informatics

Datum zadání výzkumného úkolu: 15. 10. 2021

Termín odevzdání výzkumného úkolu: 31. 8. 2022

V Praze dne 15. 10. 2021



.....
vedoucí práce

.....
vedoucí katedry

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracovala samostatně a použila jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne ....

Anna Gruberová

Poděkování

.

Anna Gruberová

Název práce:	**Optické rozpoznávání znaků na naskenovaných historických plakátech pomocí nejmodernějších metod**
Autor:	Anna Gruberová
Obor:	Aplikace informatiky v přírodních vědách
Druh práce:	Výzkumný úkol
Vedoucí práce:	Ing. Adam Novozámský, Ph.D. Computer Vision Lab, Institute of Visual Computing & Human-Centered Technology, TU Wien - Faculty of Informatics
Konzultant:	–

Abstrakt: Optické rozpoznávání znaků z obrazových dat je žádanou úlohou v dnešním světě, jelikož pro člověka je již nemožné bez automatizace zpracovat velké množství obrazových dat. Vídeňská knihovna vlastní nad 350 tisíc digitalizovaných historických plakátů, z nichž je třeba extrahovat zobrazený text. Cílem této práce je zmapovat vybrané existující metody rozpoznávání textu a porovnat je na základě testů na vybraných datasetech.

Klíčová slova: Optické rozpoznávání znaků, rozpoznávání textu, automatizace

Title:	**Optical Character Recognition on Scanned Historical Posters Using the State-of-the-Art Methods**
Author:	Anna Gruberová

Abstract: Optical character recognition from image data is a demanding task in today's world, because it is already impossible for humans to process a large amount of image data without automation. The Vienna Library owns over 350,000 digitized historical posters, from which the displayed text must be extracted. The aim of this work is to map selected existing text recognition methods and compare them based on tests on selected datasets.

Key words: Optical Character Recognition, Text Recognition, Automation

Contents

Introduction

This paper is aimed at optical character recognition and focuses on comparison of different methods that are used in this branch of image processing. The goal of the paper is to test selected methods with various parameters on different datasets. One of the datasets includes historical posters from Vienna City Library. This dataset needed to be selected from a broad database of posters and manually labeled.

In the first chapter I introduced terms used in optical image recognition and described main tasks – text detection and recognition. This is followed by a brief description of the structure of text image data accompanied by commonly used dataset examples.

The next chapter is dedicated to neural networks, which play a leading role in image recognition and in reading text tasks. I proceed from the basic network architecture to more complex networks that were developed for image and text recognition.

In the third chapter is a description of four selected methods I had chosen as methods to be tested for the purpose of this paper.

The last chapter is about performed experiments. It includes information of selected datasets, a description of implemented functions and above all a discussion about results obtained in the experiments.

In the appendix are image examples of predictions and ground truth of selected historical posters.

Chapter 1

Optical Character Recognition

Optical character recognition (OCR) is a branch of digital image processing. Its aim is to detect and convert a text on an image into a machine-readable text. This discipline can be divided into three similar, yet different tasks: reading text on scanned printed documents, reading of handwritten texts and scene text recognition (also called text in the wild). The first task is very well developed, first successful results date back to the second half twentieth and were used in commercial sector [24]. If we assume the handwritten text is on scanned single colored paper or created using a digital pen and that it was written legibly and without omitting letters in words due to fast writing, then recognition is similar to printed documents. The last task – scene text recognition (STR) is the most challenging one. The main factors that make STR a more difficult task are listed below.[6, 26]

- Complex background: in scanned documents background is white and without a distinctive pattern (omitting lines is an easy preprocessing task), while in scene images there are objects that can be mistaken for letters.
- Text diversity: text can appear in various colors, fonts, sizes or orientations.
- Distortions: photographs often suffer from noise due to bad illumination, also from motion or out-of-focus blurring, perspective distortion due to the capturing angle. Other problems come from the insufficient resolution that might be set on the camera.

Apart from these main categories of image text data there exist born-digital images (e. g. web advertisements or any cases where text was digitally added on images or videos) and poster/newspaper images. These share with scene images the diversity of text but are free from visual distortions caused by cameras. Examples of different image data are in sections 1.3 and 4.1.

Digital reading of a text on an image consists of two main tasks – text detection and text recognition. Both processes are described in the next sections. The structure can be seen in the Fig. 1.1.

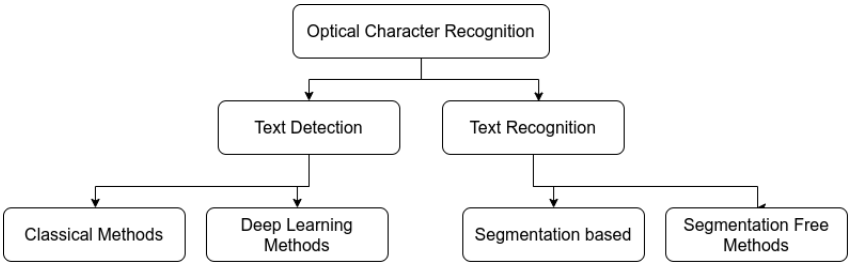


Figure 1.1: The structure of OCR tasks.

1.1 Text Detection

The first phase of digital text processing is to detect the text regions on the image. The goal is to determine a bounding box surrounding a group of letters – usually one word, but it can be also a text line consisting of few words. This box may be a bounding rectangle or a polygon which is more accurate for curved or skewed text. It is ideal when the bounding box contains the letters with as little background as possible. The methods store this information as a set of coordinates which unambiguously define the box, some methods produce cropped images based on the coordinates or a binary image with highlighted detected area.Methods used for text detection can be categorizes into formerly used classical machine learning methods and deep learning methods.

Classical Methods

Classical methods include connected component methods and sliding window methods. In the latter method an image is scanned with a moving window of certain size. In each position of the window features are computed, for example, standard deviation or histogram of oriented gradients and are compared with values known from training images with text. This approach does not have a good response for scene images, because many objects can be misunderstood as letters and vice versa. Connected component based methods extract from the image features like color, texture, edges or corners. Then they are classified either as text or non-text by a traditional classifier such as support vector machines, nearest neighbor or random forest. Detected letters are then combine into words or text lines if desired [26]. Classical methods are generally not very successful with scene images where most of the observed values are negatively influenced by distorting factors listed above. In recent years the development of neural networks enabled a new, efficient way of detecting text in images.

Deep Learning Methods

Deep learning methods are faster, more precise than the classical methods. They are automated and need less of human assistance therefore they can process bigger amounts of data. Another advantage is that algorithms can be generalized and used

for other image detection tasks. Most of the state-of-the-art methods utilize convolutional neural network (CNN). Deep learning methods can be split into bounding-box regression based, segmentation based and hybrid methods according to the survey of Raisi et al.[26].

Bounding-box regression based methods treat text as an object and predict directly the bounding box around text. However, the bounding boxes have different aspect ratio than typical objects, because text is usually long and thin. Some methods decompose the text on smaller units and concentrate on distances between the units. These methods are hard to tune during training an usually fail on distinctly curved text. Examples of bounding-box methods are TextBoxes [20], TextBoxes++ [19] or EAST [38]. Segmentation based methods investigate the image at pixel level. The image data are processed by a CNN which produces a segmentation map from which a bounding box is generated. An example of this method is PSENet [36]. Hybrid methods combine the features obtained by regression and the segmentation map from CNN. The segmentation methods tend to return false positive detection. They select pixels that are not text at the borders of letters or when there is a complicated background. Hybrid methods further process the results from segmentation and precisely detect text. Example methods are PMTD [21].

A popular method in recent years is Character Region Awareness for Text Detection – CRAFT. This method in contrast to the previously mentioned methods detects text based on character level rather than word (group of characters) level. CRAFT trains a CNN which produces two results for a character – a region score and an affinity score. Because this method goes over individual characters it performs very well on curved and deformed text and outperforms 15 popular detection methods according to the authors of CRAFT in their paper [4].

1.2 Text Recognition

Text recognition converts a detected text on an image to a string. Analogous to detection methods again recognition methods can be divided to classical methods and deep learning methods. Traditional methods work with image features such as histogram of gradients, features from SIFT[1], which are then classified by SVM or nearest neighbor algorithms. Moreover methods can be divided on segmentation based and segmentation free methods. The former classify single characters gradually and then connect them into a word. The latter examine the text line as a whole, so it can use word neighborhood for contextual information. In the following subsection four stages of segmentation free methods are described.[6, 26]

Segmentation Free Methods

Segmentation free methods follow a pipeline with four stages: image preprocessing, feature representation, sequence modeling and prediction.

[1]Scale-Invariant Feature Transform

1. **Image Preprocessing Stage**: during this phase visual quality of the image is improved as much as possible. Because in scene images background is usually not a plain color, but a complex mixture of colors in various shape, the effort is to replace it and create optimally a binary image with one color for the foreground text and another for the background. It can be achieved by using neural networks. If complete background removal is not easily achievable, basic image preprocessing such as noise removal are performed at least. Optional enhancement for scene images is to increase readability via superresolution, this removes noise caused by low resolution of the original image. If the text is perspectively distorted or curved there is an effort to straighten the text, this process is called rectification. It is a computationally expensive process, therefore it is not used very often.[6]

2. **Feature Representation Stage**: now it is necessary to extract features from the image data that are used as a representation of objects (text) in the images. For this purpose CNNs are widely used. For example namely these types VGGNet, ResNet, DenseNet, recurrent CNN (RCNN).

3. **Sequence Modeling Stage**: An optional step between final prediction and features. A contextual information is obtained via a recurrent neural network. Mostly one of two types called long short-term memory (LSTM) and bidirectional long short-term memory (BLSTM) is applied. The information is utilized during the last stage for predicting subsequent characters rather then predicting each character individually.

4. **Prediction**: in this last stage the model returns a target word or text line that was recognized. Two major technologies are connectionist temporal classification (CTC) and the attention mechanism. Because some letters span more than others it may happen that the letter is classified multiple times, to avoid this CTC introduces a blank symbol and inserts it among characters in various locations and creates character sequences. CTC then trains the network to maximize the probability distribution over all possible sequences. This approach is to a great extent dependent on a lexicon or language information about word formation. Attention-based methods utilizes RNNs [18]. "The attention mechanism learns the alignment between the input instance image and the output text sequences by referring to the history of the target characters and the encoded feature vectors" [6, page 12].

1.3 Datasets

Optical Character Recognition requires data as any other machine learning task. Data are usually divided in two main types - scene and synthetic. Scene datasets contains photographs of real world objects and sceneries where some text occurs, for example shop signs, road signs or car plates. Synthetic dataset are automatically generated images where words are chosen from a extensive dictionary, a font is picked for each word and some sort of deformation is applied. It can be a text distortion to

make the text curved or projectively altered, as well as blurring or lighting changes that make the text less obvious for the detector.

Synthetic datasets are usually used for training the models, because it is easier to generate millions of synthetic images rather than to take even one hundredth of a such number of photographs. Needless to say that when generating an image, the ground truth is known and can be saved during the generation process while photographs have to be manually labeled which takes time and might be inaccurate or automatically labeled which also often leads to many mistakes. Scene datasets are then used for testing purposes or for fine tuning a pretrained model.

In this paragraph sample datasets are introduced. To begin with synthetic dataset MJSynth is a very important dataset because it consists of almost 9 million images covering 90,000 English words. It includes data only for recognition which means that one image has only one word and border of the image represents the word bounding rectangle.[23]. Another synthetic dataset is called SynthText and contains 800 thousand images with approximately 8 million word instances written on the images [33]. Three scene text datasets were created for International Conference on Document Analysis and Recognition (ICDAR) competition. Sets ICDAR03, ICDAR13 and ICDAR15 were used in competitions in years 2003, 2013 and 2015, respectively. First two ICDAR datasets include only horizontal text, text of various orientation appears in set from 2015 [26]. Another widely used dataset is The Street View Text (SVT) which contains images with text harvested from Google Street View [32]. SVT and most other scene text datasets offers mainly frontal text with minimal perspective distortion. However, perspective text is frequent in real life applications of OCR for example previously mentioned street photographs, where it is impossible to capture every visible text from frontal view. Thus Phan et. al [25] created a new StreetViewText-Perspective derived from SVT, it shows the same places as SVT but from different perspective. Another dataset CUT80 focuses on curved text as well as CTW1500 dataset. For text recognition there exist for example IIIT5k dataset containing 5000 cropped images harvested from Google image search. It combines both scene text images and born-digital images [14]. One of the most widely used dataset is COCO-Text which includes over 60,000 images with almost 250,000 word instances [7].

Datasets that were used for comparison of detection and recognition methods, namely, SCUT-CTW1500 dataset, Kaist Scene Text Database, Born-Digital Images and the historical poster dataset are described in more detail in Chapter 4.

1.4 Evaluation

Text Detection

A detection model tries to detect a text regions as precise as possible. It predicts a bounding box around the text and return its coordinates. As said earlier it can be an irregular polygon or a rectangle.

For measuring the performance of a detection model in general object detection a intersection over union (IoU) is used and it can be evenly applied on text detection. The principle is to compute the intersection of a ground truth bounding box and predicted bounding box and divide it by union of those two areas.[28]

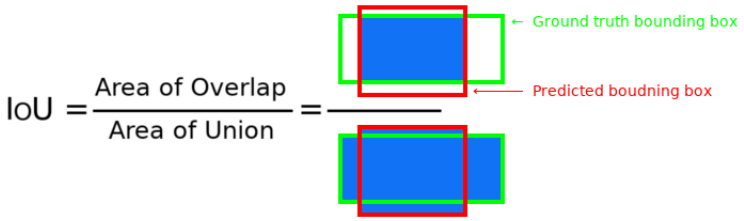


Figure 1.2: IoU formula with a bounding box visual interpretation.

Text Recognition

To test how much is the recognition model successful various evaluation metrics were created. The simplest that comes to a mind is compare the word in the image with the predicted one. We can either compare words strictly and as correct classify only when all characters match and even a small mistake makes the prediction false, or we can use the metric called word error rate (WER). Before we define how it is computed, we must introduce three types of error that are taken into account:

- **substitutions**: words with one or more misspelled characters,
- **insertions**: words that were added (do not appear in original text),
- **deletions**: missing words.

WER is then defined as follows.

$$WER = \frac{(i_w + s_w + d_w)}{n_w}, \tag{1.1}$$

where i_w is the number of inserted words, s_w is the number of altered words, d_w is the number of lost words and n_w is the number of all words in a ground truth text. The chosen value of WER is the one where the sum in numerator is minimal.[5]

This approach is straightforward and it is a useful for long text, for example scanned books or documents. However, used on scene text or generally images with sparse text that does not contain sentences, it has some drawbacks. Scene text images do not have a long contextual information, there are words that belong together (such as names of countries) but mostly there are street signs, telephone numbers, names of people or companies that do not occur in lexicon. In these cases it is often that the recognizer makes mistake in just one character, that can be barely legible due to an extravagant font or lighting conditions or other distortions. Therefore, it is better to focus on characters and validate each letter in a word

separately. For this purpose is used, similar to word error rate, a character error rate (CER). There are again three possible error types – substitution, insertion and deletion of a character in a word.

The formula for CER computations is also analogous

$$CER = \frac{(i + s + d)}{n}, \tag{1.2}$$

where i is the number of inserted characters, s is the number of substituted characters, d is the number of deleted characters and n is the number of all characters in a ground truth word.[5]

The sum of the three error operations is called a Levenshtein distance. Let s be the ground truth (source) string and t the predicted (target) string. The steps of the algorithm are [12]:

1. Let n be the length of $\boldsymbol{s}$ and m the length of $\boldsymbol{t}$. Construct an empty matrix $\boldsymbol{d}$ with $0, \ldots, n$ columns and $0, \ldots, m$ rows.
2. Initialize the first row to $0, \ldots, n$ and first column to $0, \ldots, m$.
3. Go through $\boldsymbol{s}$ $(i = 1, \ldots, n)$ and $\boldsymbol{t}(j = 1, \ldots, m)$.
   - Compare $\boldsymbol{s}[i]$ and $\boldsymbol{t}[j]$:
     if it equals, set $cost = 0$,
     if it does not, set $cost = 1$.
   - Set $\boldsymbol{d}[i, j]$ equal to the minimum value of:
     $\boldsymbol{d}[i-1, j] + 1$ (deletion),
     $\boldsymbol{d}[i, j-1] + 1$ (insertion),
     $\boldsymbol{d}[i-1, j-1] + cost$ (substitution).
4. Repeat step 3 until $\boldsymbol{d}[m, n]$ value is computed. That is the Levenshtein distance.

It is clear that the number of mistakes can exceed the length of the source word, which leads to error rate larger than one hundred percent. For this purpose sometimes normalized CER is used. To obtain it the sum of errors is divided not by the number of characters in the source word but by the sum $(i+s+d+c)$, where i, s, d are the error operations and c is the number of correct characters the predicted word and can be computed as $c = n - d - s$. This way the denominator is always larger than the numerator (or equal). After modifications we get the following formula:

$$CER = \frac{i + s + d}{n + i} = \frac{Levenshtein\ distance}{n + i}. \tag{1.3}$$

Chapter 2

Neural Networks

A perceptron unit is able to find a linear boundary between two separable classes. In n-dimensional space we talk about a separating hyperplane. The following equation

$$\boldsymbol{w}^\top \boldsymbol{x} + w_{n+1} = 0 \tag{2.1}$$

is a vector form of the hyperplane equation, where $\boldsymbol{w}$ is a weight, n-dimensional column vector, $\boldsymbol{x}$ is also n-dimensional vector and it contains the coordinates of a point, which is being classified, w_{n+1} is a bias. We can simplify the notation by creating a *n+1*-dimensional vectors: $\boldsymbol{x} = [x_1, \ldots, x_n, 1]^\top$ and $\boldsymbol{w} = [w_1, \ldots, w_n, w_{n+1}]^\top$. We want to find a weight coefficients that satisfies the following property:

$$\boldsymbol{w}^\top \boldsymbol{x} > 0 \qquad \boldsymbol{x} \in class_1 \tag{2.2}$$

$$\boldsymbol{w}^\top \boldsymbol{x} < 0 \qquad \boldsymbol{x} \in class_2 \tag{2.3}$$

Then the perceptron learning algortihm can be described as follows:

For any $\boldsymbol{x}(k)$, at step k:

1. If $\boldsymbol{x}(k) \in class_1$ and $\boldsymbol{w}^\top \boldsymbol{x} \leq 0$, let

$$\boldsymbol{w}(k+1) = \boldsymbol{w}(k) + \alpha \boldsymbol{x}(k) \tag{2.4}$$

2. If $\boldsymbol{x}(k) \in class_2$ and $\boldsymbol{w}^\top \boldsymbol{x} \geq 0$, let

$$\boldsymbol{w}(k+1) = \boldsymbol{w}(k) - \alpha \boldsymbol{x}(k) \tag{2.5}$$

3. Otherwise, let

$$\boldsymbol{w}(k+1) = \boldsymbol{w}(k) \tag{2.6}$$

where $\boldsymbol{w}(1)$ is arbitrary and $\alpha > 0$ is a constant called learning rate. Sum of the products, $\sum_{k=1}^{n} w_k x + w_{n+1}$ is then passed through an activation function. In case of perceptron it is a threshold function returning either 1, when $\boldsymbol{x}$ belongs to $class_1$,

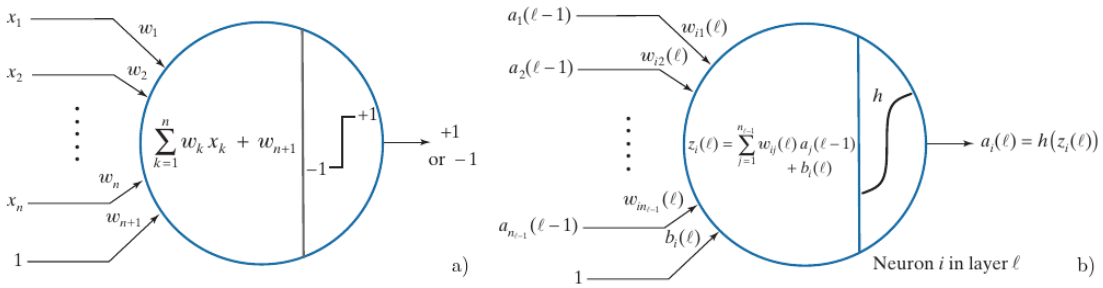


Figure 2.1: Model of a perceptron unit (a), model of an artificial neuron (b) with used operations. The letter h denotes an activation function, l denotes a particular layer in a multilayer network. The element on the right is sometimes called a sigmoid neuron.[11]

or -1 when it belongs to the second class $class_2$. In Figure 2.1.a. is shown a model of a perceptron.[26]

Linearly separable data are rather rare in real life problems. One possibilities is to use more perceptron units, however, the solution comes with neural networks and computing elements called artificial neurons. These elements are similar to perceptrons as they perform the same computations, but have a different attitude to processing the results. Schematics of a perceptron unit and an artificial neuron can be comapred in figure 2.1. The perceptron activation function is very insensitive to small signals which can lead to false results. If the activation function is changed from a hard threshold to smooth function, results are then handled more carefully. There are few commonly used smooth activation functions, such as sigmoid, hyperbolic tangent or ReLu (rectifier linear unit) function. In Figure 2.2 are the equations and shapes of the mentioned functions.[11]

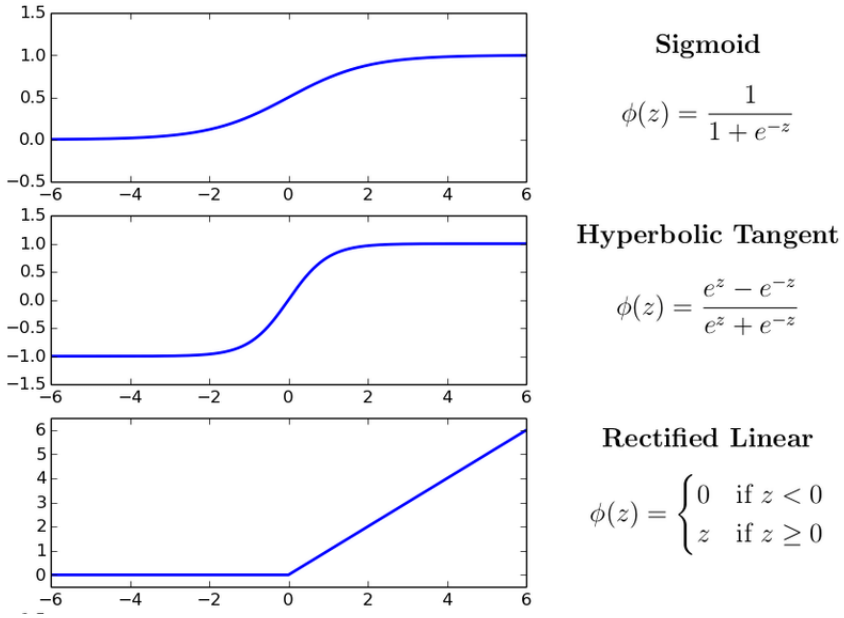


Figure 2.2: Commonly used activation functions in neural networks.[13, altered]

A generic neural network is depicted in Figure 2.3. By layer we understand a group of neurons, usually symbolized in a column. First layer contains the input vector $\boldsymbol{x}$, then every other layer contains the activation values of neurons in this layer. The conntecting lines between each two neurons signifies a fully connected neural network, where output of every neuron from one layer is used as input for neurons in the following layer. Values of initial layer are known and also are the last output values, all neurons (and layers) between first and last layer are therefore called hidden. When there are more then one hidden layer we talk about deep neural network. Usually the number of output neurons is equal to the number of observed classes. For the rest of this chapter we will assume that there are no loops in the network.[11]

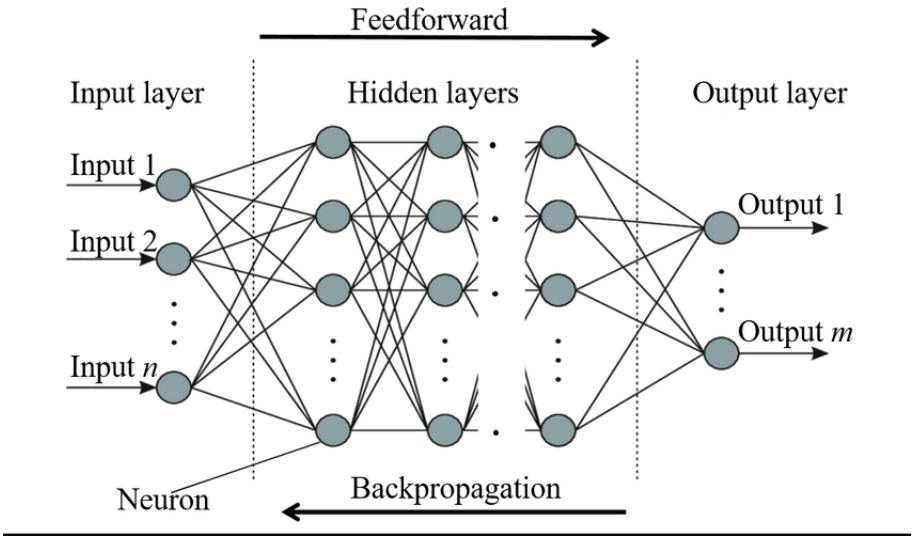


Figure 2.3: Fully connected neural network with processes.[1]

A forward pass through neural network maps the values of vector $\boldsymbol{x}$ (the input layer) to the output layer, thus to determine the class of the vector $\boldsymbol{x}$. Steps of a forward pass can be written in matrix notation. This aproach is good for computing simultaneously with multiple input vectors and all neurons in one layer. The forward pass can be described in three steps:

1. Input
$$\boldsymbol{A}(1) = \boldsymbol{X} \tag{2.7}$$

2. Feedforward step
$$\text{For } l = 2, \ldots, L \qquad \boldsymbol{Z}(l) = \boldsymbol{W}(l)\boldsymbol{A}(l-1) + \boldsymbol{B}(l),\ \boldsymbol{A}(l) = h(\boldsymbol{Z}(l)) \tag{2.8}$$

3. Output
$$\boldsymbol{A}(L) = h(\boldsymbol{Z}(L)) \tag{2.9}$$

Matrix $\boldsymbol{W}(l)$ contains all weight vectors of all nodes in one layer l, $\boldsymbol{X}$ contains input vectors, $\boldsymbol{A}(l)$ contains output values from layer l, $\boldsymbol{B}$ is the matrix of biases,

$\boldsymbol{Z}(l)$ contains the net inputs to neurons in layer l, h is an activation function. This process of predicting works when we know the right values of weights and biases. These can be obtained by training a neural network via backpropagation.[11]

During training of neural network we work with data where it is known for each sample to which class it belongs. This means that we know the values of all otuput neurons in the net. However, we do not know the values of outputs of hidden neurons. A process called backpropagation is used to obtain information about hidden neurons. It can be devided into four steps. These steps are repeated until the value of cost function is reduced to a desired level. One repetition is called an epoch, thus the number of iterations is the number of epochs used for training the network.[11]

1. Input of data from training set.

$$\boldsymbol{A}(1) = \boldsymbol{X} \tag{2.10}$$

2. A forward pass to classify the data into class and determine the error of misidentified classes (sometimes this error function is called cost function) based on ground thruth from the training data.

$$\begin{aligned} \text{For } l = 2, \ldots, L \qquad & \boldsymbol{Z}(l) = \boldsymbol{W}(l)\boldsymbol{A}(l-1) + \boldsymbol{B}(l), \\ & \boldsymbol{A}(l) = h(\boldsymbol{Z}(l)), \\ & \boldsymbol{D}(L) = (\boldsymbol{A}(L) - \boldsymbol{R}) \odot h'(\boldsymbol{Z}(L)) \end{aligned} \tag{2.11}$$

3. A backward pass that sends the output error back through the network, where changes to update neuron paramters are computed.

$$\text{For } l = L-1, L-2, \ldots, 2 \qquad \boldsymbol{D}(l) = (\boldsymbol{W}^\top(l+1)\boldsymbol{D}(l+1) \odot h'(\boldsymbol{Z}(l)) \tag{2.12}$$

4. An update of weights and biases of neurons.

$$\begin{aligned} \text{For } l = 2, \ldots, L \qquad & \boldsymbol{W}(l) = \boldsymbol{W}(l) - \alpha \boldsymbol{D}(l)\boldsymbol{A}^t op(l-1), \\ & \boldsymbol{b}(l) = \boldsymbol{b}(l) - \alpha \sum_{k=1}^{n_p} \delta_k(l), \\ & \boldsymbol{D}(L) = (\boldsymbol{A}(L) - \boldsymbol{R}) \odot h'(\boldsymbol{Z}(L)), \\ & \boldsymbol{B}(l) \text{ consist of horizontally stacked vectors } \boldsymbol{b}(l) \\ & \text{for } n_p \text{ times}, \\ & \boldsymbol{\delta}_k(l) \text{ are the columns of matrix } \boldsymbol{D}(l), \end{aligned} \tag{2.13}$$

where $\boldsymbol{W}(l)$ is the matrix of weights of all nodes in one layer l for the inputs from $\boldsymbol{X}$, which contains multiple input vectors, $\boldsymbol{A}(l)$ contains output activation values from layer l, $\boldsymbol{B}$ are the biases, $\boldsymbol{Z}(l)$ contains the net inputs to neurons in layer l, $\boldsymbol{\delta}(l)$ tells us the rate of error change with respect to a change in the net input to any neuron in the network $\delta_k(l) =$, α is the learning rate used in training, h is an activation

function. $\boldsymbol{W}(1)$ and $\boldsymbol{B}(1)$ are set as random small numbers when initializing the process.[11]

The error function for all output neurons for a single input $\boldsymbol{x}$ is defined as

$$E = \sum_{j=1}^{n_L} E_j = \frac{1}{2}\sum_{j=1}^{n_L}(r_j - a_j(L)) = \frac{1}{2}||\boldsymbol{r} - \boldsymbol{a}(L)||^2 \tag{2.14}$$

where $\boldsymbol{r}$ is a desired response for a given input $\boldsymbol{x}$, $\boldsymbol{a}(L)$ is the output of last layer in the network, in the last term is used the notation of the Euclidan vector norm. The error for all input vectors[1] is equal to the sum of the individual errors.[11]

2.1 Convolutional Neural Networks

The procedures described in the previous part dealt only with the case where the input data are in the form of a vector. In optical character recognition we work with image data that are not primarily represented as vector but as a matrix of pixel values. The matrix can be linearized from 2D to 1D when indeces are mapped gradually. However, this approach does not consider spatial relationships that may be present among specific pixels. For examples edge or color similarities which are significant in text detection. Convolutional neural network (CNN) accepts 2D matrix as input and extracts features from the given image, these features are then fed to a classic fully connected neural network. A diagram describing a simple CNN is in Figure 2.4. We will discuss individual steps visible in this figure below.[11]

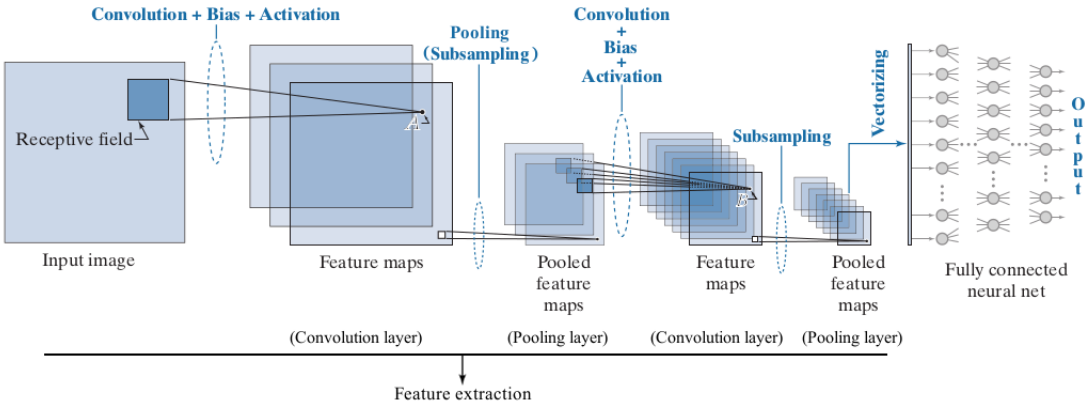


Figure 2.4: A simple CNN with LeNet architecture.[11, altered]

First, a region of of pixels of the input image, a receptive field, is selected. This field is moved over the image with a certain step. At each position a convolution is performed and values are stored in a 2D matrix. The size of the step determines the size of the resulting matrix (for example step of size two reduces the resolution of the image by one half). A mathematical definition of convolution operation is written in Definition 1. To each value obtained by convolution a bias is added and then

[1] a total network output error

is passed through an activation function. The new matrix thus obtained is called feature map. A single feature map is generated with same weights in convolution kernel and same bias, because this way a same feature is detected through the image. For another feature map different weight and bias is used. A group of feature maps in a CNN are called a convolutional layer. After the convolution step pooling is performed (sometimes called subsampling).[11]

Definition 1 *Let $f(x, y)$ be in image and w a kernel of size $m \times n$. Convolution with kernel w is denoted as $(w \star f)$ and defined as*

$$(w \star f)(x, y) = \sum_{u=-k}^{k} \sum_{v=-l}^{l} w(u, v) f(x - u, y - v). \tag{2.15}$$

Pooling is responsible for reduction of output dimension and causes a translational invariance. It is done by dividing a feature map into 2×2 adjacent (non-overlapping) regions, the four values in this region are replaced by only one value. Common pooling methods are: average pooling, the values in region are replaced by the average of the values; max pooling, the values are replaced by the maximal value from the region; L_2 pooling, the final value is obtained by computing the Euclidan norm of the four values. We obtain one matrix for each feature map from the convolution step. The bunch of matrices from the pooling step is called a pooling layer. The result of the last pooling layer is vectorized and sent to the fully connected neural network. The training procedure of the CNN is analogical to the training of a fully connected neural network. However, convolution is used instead of matrix multiplication and the output from fully connected network has to be converted into 2D matrix.[11]

VGG16 is an example of a CNN used in OCR. The basic configuration consists of 16 weight layers and uses a 3×3 receptive field for convolution. The output dimension is reduced by max pooling.[35]

2.2 Recurrent Neural Networks

Recurrent neural networks (RNN) same as feedforward networks has an input, hidden and output layer. Unlike the classic networks RNNs share parameters (weights and biases) across each layer of the network and they remember results of computations. This approach is very useful in contextual tasks such as language problems (speech and written text recognition) where the position of a word in a sentence and surrounding words can help predict text.

During training of RNN errors from output to input layer are calculated but unlike in standard backpropagation the errors are summed up because of the shared parameters. This process makes RNNs prone to two problems called exploding and vanishing gradients. The former one happens when gradients are large and by summing they enlarge too much to be represented as undefined (NaN) value, which leads

to instability of the model. The latter refers to the case when gradients are contrarily too small, they continue to decrease and the weights become insignificant.[9]

Long short-term memory

Long short-term memory (LSTM) is a type of RNN. It contains so-called memory blocks, which are memory blocks (cells) located in the hidden recurrent layer. Each memory block contains three gates – input, output and forget gate. The input one controls the flow of information in to the cell. The output gate controls what information is to be passed to the rest of the network. The forget gate was added later and it manages the amount of information stored in cell before new information is received. Generally the cell input and cell output activation functions is tanh and the network activation function is softmax. A scheme of LSTM is in Figure 2.5.[29]

LSTM network has some variations. It is possible to stack LSTM layers on top of each other between the input and output layer and create a deep LSTM. Another type is a bidirectional LSTM (BLSTM) which differs from normal LSTM that it remembers not only information from the past but also from the future (e. g. when the model wants to predict a word in a sentence, it knows all the words behind the currently predicted word and also all the following words). It enables better usage of contextual information which is crucial in language processing.[29, 39]

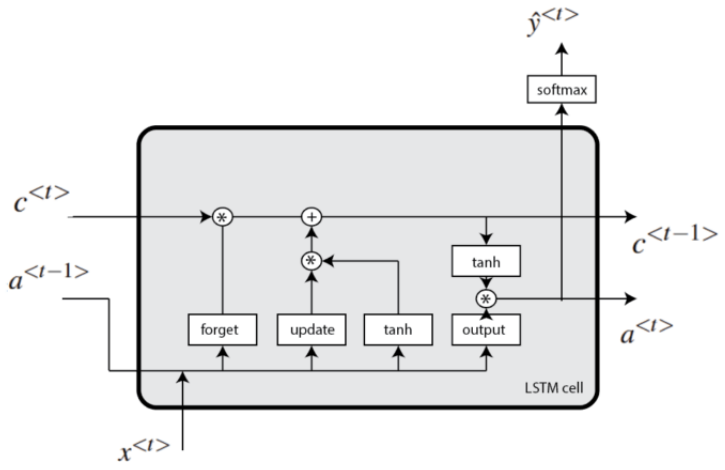


Figure 2.5: Scheme of a LSTM recurrent neural network.[39]

2.3 Convolutional Recurrent Neural Networks

Convolutional recurrent neural network is a type of a neural network designed for the purposes of image object recognition. It was introduced by Shi and al. [30] in the year 2015. It combines the advantages of a deep CNN and RNN. It reads directly features from image data without prior feature extraction or image preprocessing such as binarization as CNN does. It produces a sequence of labels as RNN. Further it has no restriction on the length of an object (texts are rather long and flat), it

can learn from words so there is no need to segment text to individual characters. CRNN also has a smaller amount of parameters than a deep CNN thus it requires less storage space.[30]

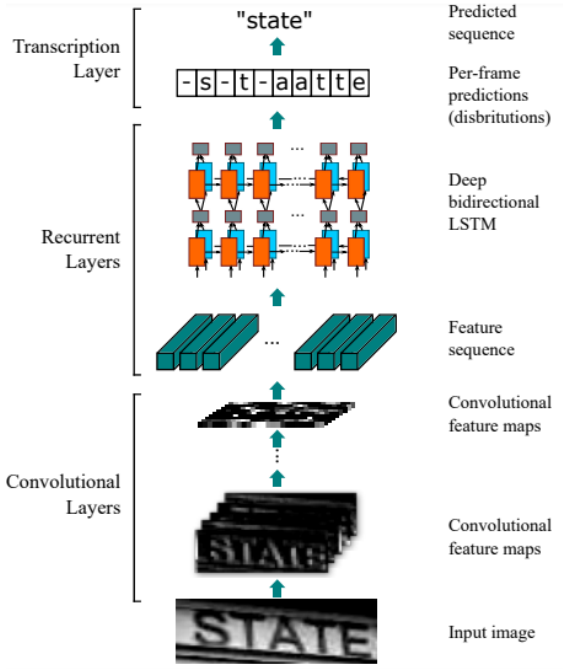


Figure 2.6: An architecture of CRNN for recognizing objects in images.[30]

First CNN with max pooling extracts features directly from the image input. The convolutional layers are followed by recurrent ones, which take the frame output of CNN and predict labels for each frame, then a transcription layer translates the segmented labels into a continuous text. This architecture is visible in Figure 2.6.

The images, that are to be recognized, are scaled to the same height and then feature map is obtained by CNN. Feature vectors are extracted from each column of the feature map, so a vector covers a rectangle area of the input image. This are is called a receptive field. CRNN uses then a deep bidirectional LSTM network to predict labels. During transcription the model picks the label of a frame which has the highest probability. This can be made either with a help of a lexicon or without it.[30]

Chapter 3

OCR tools

OCR software can be divided in four main types by its functionality - text detector, text recognizer and full OCR engine. The last one can either be an end-to-end system, which does both detection and recognition together which means that these two processes influence each other, or a tool that combines a separate detector and recognizer. For an end user who needs to convert text from an image into a computer readable format. The supply of such tools is wide, ranging from open source libraries for various programming languages to commercial softwares with modern GUI. New methods are still being developed as there is always space for improvement. New methods can come from commercial background or are developed for international OCR competitions. In the next sections a selection of the free available tools is described.

3.1 CRAFT

Character Region Awareness for Text Detection (CRAFT) is framework for scene text detection introduced by Clova AI research group. It uses a Convolutional Neural Network. It performs well also on curved or differently deformed texts. Its methodology is to localize individual characters then characters belonging to the same word (based on distance) can be connected into word box or polygon. After that bounding box is created around it and output contains the rectangle coordinates.[4]

CRAFT uses a fully convolutional neural network based on VGG-16.The network returns two values – an affinity and region score. "The region score represents the probability that the given pixel is the center of the character and the affinity score represents the center probability of the space between adjacent characters" [4, page 3]. Because CRAFT detect individual characters it was necessary for the authors to create ground truth with bounding boxes for each character. Both scores, thus the probabilities they represent, are encoded with a Gaussian heatmap which is often used were ground truth regions are not strictly bounded. The model was trained on a synthetic dataset SynthText for 50k iterations, further weakly-supervised training was performed on ICDAR datasets.[4]

This method has similarly successful performance as other state-of-the-art methods and in last years it is often used as a detection tool in systems that recognize text from unsegmented images. There exists an official Pytorch implementation of CRAFT for python which can be cloned from Clova AI GitHub repository[1], however it is not a python package. A package for python of CRAFT detector exists under the name `craft-text-detector`.

3.2 Tesseract

Tesseract is an open source text recognition engine. It supports over 160 languages can be trained to recognize new ones. Originally Tesseract was created by Hewlett-Packard in late 1980s, from 2006 it is developed and maintained by Google. As it does not have a built-in GUI direct use is via command line. However, there exist a significant number of GUIs for Linux, Windows, Mac for computer usage and also for Android and iOS to use on mobile phones and few online OCR services. Another way how to use the engine is via libraries for computer languages, namely for example they exist for Java called tess4j, python called pytesseract, R, Ruby and others. [34]

Tesseract is mainly used as tool for recognizing documents (with both computer font text or hadwritten text). Best results are obtained on preprocessed images. The preprocessing includes noise reduction, horizontal alignment of text, elimination of dark borders around text region, conversion to binary black and white picture and other adjustments depending on the nature of the picture. The ideal image for Tesseract is a legible, typed, black text on plain white. Thus when used on scene text images it gives generally worse results than other OCR softwares.

Computations with Tesseract are supported for GPU and also CPU. Since version 4, that was announced in 2018, Tesseract uses for recognition Long Short Term Memory (LSTM) model (kind of RNN). A simple pipeline of Tesseract is in Figure 3.1. First a binary image is created from the input one, then characters are found with the connected components algorithm. Joined characters are then chopped and broken ones are connected, now each character should be separate. Then characters are recognized and further joined into words. The words are verified in a lexicon and a word with the smallest distance is selected.[31]

By default Tesseract expects a page of text – black letters on white background grouped in horizontal lines, where font type and font size vary only slightly. To deal with differently distributed text over an image Tesseract provides thirteen page segmentation models (PSM). When selecting the right model Tesseract performance can increase from zero up to almost perfect results. Description of all the PSMs can be find directly via Tesseract help command in console application. Thanks to the various PSM Tesseract works also as a detection tool but often happens that during a search for text in scene images it mistakes objects and structers for text. Such example can be seen in Figure 3.2. Besides the PSM parameter user can set also

[1] https://github.com/clovaai/CRAFT-pytorch

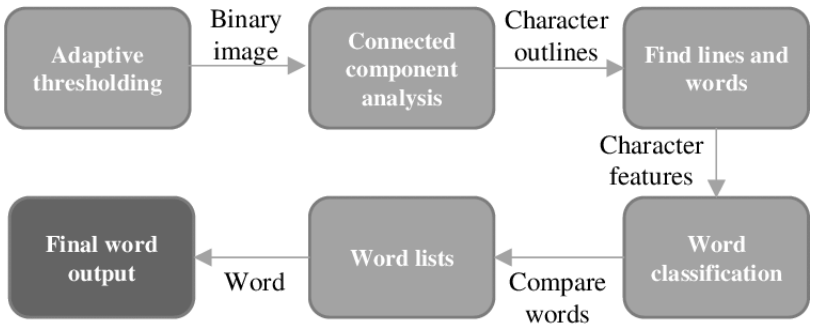


Figure 3.1: Pipeline architecture of Tesseract OCR engine. [10]

OCR engine modes (OEM). There are four settings differing in whether a LSTM network should be used.

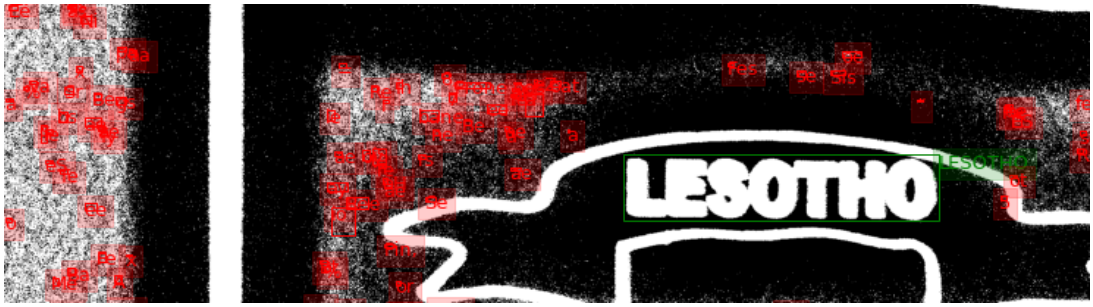


Figure 3.2: Tesseract mistakes background texture for text. Red rectangles and characters denotes predicted words. Part of an preprocessed image from CTW1500 dataset.

3.3 EasyOCR

EasyOCR is a product of Jaded AI for both image text detection and recognition. it supports over 80 languages and various scripts such as Latin, Chinese, Arabic etc. The company offers software with web interface for free and also prepaid version which enables usage of a new model for custom data. However, in addition to the web interface, the company also created a python package under the same name.[2]

The product is still in development and aims for wider functionality. A future idea of EasyOCR package is to provide an easy-to-use tool where one can plug-in already created state-of-the-art models and use them for annotating. Pipeline of EasyOCR behavior is shown in the image 3.3. As it can be seen in this image, default detection model is CRAFT and for recognition is used CRNN (Convolutional Recurrent Neural Network)[2]. The implementation of this network is composed of following components: feature extraction (Resnet is used) and VGG (Convolutional Neural Network), sequence labeling (BLSTM is used) and decoding (CTC is used).[3]

[2]The description of this network is in Chapter 2.3.

EasyOCR package by default computes annotation on GPU, however there is a possibility for CPU computations (provided that the selected model supports it).

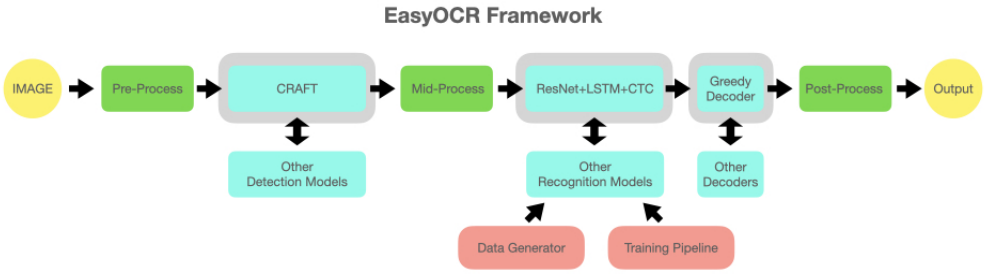


Figure 3.3: Diagram of EasyOCR pipeline. Grey slots are placeholders for models. The mentioned models are the ones used as default. [3]

3.4 Keras-ocr

keras-ocr is a python library used for detecting and recognizing text in images created by Fausto Morales. It works with variety of languages and with different writing scripts. It allows computing on CPU as well as on GPU. It unites the CRAFT text detection model and an implementation in Keras python library of CRNN for recognizing text, worth mentioning this is a different implementation of CRNN than in EasyOCR.[17]

On the official website[3] of the package there is a comparison of this method with two other OCR APIs – Google Cloud Vision and AWS Rekognition. Their performance was tested on 1,000 images from the COCO-Text validation set using a basic pretrained model of each method. None of the investigated methods performed poorly; however, AWS Rekognition had the worst precision and recall results. Google's method and keras-ocr has similar results. It is important to mention that no tuning parameters were used in any of these methods. Another candidate for comparison was Tesseract but it performed on very badly on given data, most likely due to the fact that Tesseract is suitable for scanned documents rather than for photos of real life scenery and objects with text. [17]

CRAFT already provides a pretrained model which can be used directly without modification for text detection or it is used as initial model for training a new model on new data. This model was trained on three datasets (SynthText, IC13, IC17) and supports English and multi language text detection.[27] Similarly for recognition, CRNN also has a pretrained model. This model was trained on the synthetic word dataset which consists of 9 million images with vocabulary of 90K English words.[**?**] To use these models in the keras-ocr library one either doesn't specify anything and use the defaults, or pass the value `clovaai-general` for the CRAFT pretrained model or `kurapan` for the CRNN model.

[3] `https://pypi.org/project/keras-ocr/`

Keras-ocr offers preprocessing for four public datasets though any text image dataset can be examined using this tool. These four datasets are: BornDigital dataset, COCO-Text dataset, ICDAR 2013 dataset, ICDAR 2019 dataset (only Latin-only scripts).[16]

Chapter 4

Experiments

This section is divided into two main section – one describes used datasets in the experiments, the other provides an implementation of functions I have written for the testing.

4.1 Datasets

SCUT-CTW1500 Dataset

SCUT-CTW1500 dataset contains exactly 1500 images of real-world, scene text in English language. Sample images can be seen in Figure 4.1. The key feature of this dataset is that each image contains horizontally aligned text, multioriented text and curved text. There are cases where the curvature is only slight and cases where text forms a circle with letter upside down. Recognizing multi-oriented and curved text is more of a challenge than pure horizontal text. This dataset is split to train and test data. Two thirds of dataset thus one thousand images for training and five hundred for testing. According to the description of this dataset on relevant GitHub repository dataset was manually labeled and lately corrected, therefore labels seem to be very accurate. However for example ground truth for image 1313.jpg misses all occurences of letter I, as the depicted font was probably misread.[37, 22]

The ground truth for train data are in XML format and each file carries information about the file name of respective image file, text information – i.e., words in a text line, 14 coordinates of a bounding polygon and coordinates, height and width of a circumscribed rectangle. Later the authors added coordinates of center point of each English letter to be used as detection ground truth. The ground truth of test data is in simple text file (TXT) and contains only 14 coordinates of the bounding polygon and a text which is within that region. There is a minor issue with labels that it usually contains a full text line with multiple words and coordinates are not assigned to individual words but to text region as whole. Most end-to-end system detect words rather than groups of corresponding words. This fact needs to be taken into account when evaluating results.

Figure 4.1: Sample images of SCUT-CTW1500 dataset.

KAIST Scene Text Database

This dataset contains 3000 images of photographed text. It can be divided into three major categories – text of Korean language, English language and mixed languages. As I concentrate on text in latin script in this paper further information relates to the English language dataset. The number of images is then reduced to less than four hundred images. Figure 4.2 shows few samples of this dataset. Photographed objects are mostly shop banners or parts of magazine front pages. Photographs were either taken by a high-resolution digital camera or a low-resolution mobile phone camera.[15] Each photography has a ground truth description and a bitmap image. In the bitmap file only text is highlighted (by white or red color) and everything else apart from text is set as black. Ground truth files are in XML format and includes a name of an image, its resolution and bounding box for each word and also a bounding box for each letter of the word.

To use this dataset for testing and training the XML ground truth needed to be converted to string and int values. I wrote a parser, that combines letters to form a word that is within a given bounding box. I changed the notation of bounding boxes from one coordinate, width and height attributes to two top left and bottom right coordinates. The name of the parsing function is `read_gt_kaist`.

Unfortunately this dataset has few errors in filenames of corresponding files or in the content of XML files. Usually these are only typos, however they prevent automatic preprocessing of dataset. Due to this problem these mistakes need to be found and manually corrected. Also there is a small number of ground truth XML file with fully missing data. Despite these shortcomings this dataset is useful because of the bitmap files. This allows to compare results of both images affected by shooting conditions and images dependent only on font and position.

Figure 4.2: Sample images of KAIST Scene Text Database dataset.

Born-Digital Images

Born-Digital Images contains data of images with text that can be found on various websites. Samples of this dataset can be found is in Figure 4.3. There are mostly advertisements, company logos or website headers. Such pictures cannot be classified neither as real scene dataset, neither as synthetic one. On one hand this dataset shares with scene datasets the variability in font styles and sizes, different text orientations and complex colour placement. On the other hand it differs in size because low resolution is significant in smooth and fast loading on websites. Also no noise is present due to lighting conditions. Geometrical deformations that result when capturing a real scene with camera also do not appear here. However compression to lower resolution can lead to artifacts and aliasing. In general we can say that letters are more clearly visible than in photographed text as easy readability is crucial in successful advertising.[8]

The dataset is available for download from the website of Robust Reading Competition. First version was published in 2011 and revised two years later, it contains separate dataset for text localization, segmentation and then for word recognition. In 2015 they published an end-to-end dataset with ground truth for all tasks. The dataset is split in training and testing data. However, ground truth for testing data contains only a possible vocabulary of words in images and no coordinates. This might be due to the fact that the competition might be still ongoing or there was not a sufficient demand for complete ground truth. As for training data, each image has a corresponding TXT file with coordinates of four vertices of bounding rectangle and a word. Text lines are separated and the text within rectangle is always one word. Extracting ground truth is done in the function `read_gt_bd` and unlike preceding datasets there was no parsing needed, only read the desired values from a file. Unfortunately, there are quite a few missing words, usually words that have two or less characters. This can affect the evaluation when the model finds such a short, missing word.[8]

Figure 4.3: Sample images of Born-Digital Images dataset.

Vienna City Poster Dataset

The Vienna City Library possesses a collection of 350,000 poster images. A sample of the collection was provided by the organization to the Technical University of Vienna for research purposes. It consists of 5050 images. From these images we manually labeled 257 images and created a testing dataset. Sample of the dataset is in Figure 4.4. As it can be seen from the example images, the dataset includes mainly posters with German language, though few images have words in English or Czech.

The posters have neither the characteristic of a scanned text documents, neither of scene images. They are most similar to born digital images. However for the posters is typical one thing – a part of the text is large relative to the image size and another part is tiny. The huge text is usually a brand or product name or a name of an event. The small text is the name of the author of the poster or the printer where the posters was printed. Both of these texts are a challenge for an OCR engine, because large letters are misinterpreted as objects and the height of the tiny ones is not bigger then twenty pixels and letters are often blurred. Middle-sized text also appears in the posters and is generally well recognized.

Due to the presence of the small text. All images in the dataset need to be in the original resolution. The bigger side is always 4096 pixels. This leads to a large dataset even when the number of images is less then three hundred. A single JPG image has approximately 4 Megabytes. A PNG image in the dataset has about 20 Megabytes. To reduce the final size of the dataset I decided to convert the PNG files to JPG[1]. The final dataset of JPG images is 1.1 GB large, before it was 2.8 GB.

[1]For converting the images I used the console application ImageMagick, with the command

Figure 4.4: Sample images of Vienna City Poster Dataset.

The high resolution of images makes demands on the hardware when working with the dataset and makes it impossible to run on machines with insufficiently large computing memory.

The annotation tool used for labeling the selected posters is called Aletheia from the PRiMA Research[2]. The program can be downloaded for Windows operating system, it offers both Lite and Pro version. A free one month trial is offered or for academic use the PRiMA Research gives an extended license. Using this tool we labeled every separate word or group of letters (such as poster identification number) found in an image on the selected posters. Words that were forming a text line, were afterwards labeled as a text line (single word is a text line consisting of one word). Individual characters were not labeled. Bounding boxes for both words and text lines are rectangles, even for curved text. Word bounding boxes were drawn pixel precise with the intention of no margin between the box and the word. Aletheia software produces a XML file with PRiMA Research tags, example for the image P-2151.jpg is in the code below and in Figure 4.5 is a graphical representation of this XML file.

```
<?xml version="1.0" encoding="UTF-8"?>
<PcGts xmlns="http://schema.primaresearch.org/PAGE/gts/pagecontent
    /2019-07-15" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance" xsi:schemaLocation="http://schema.primaresearch.org/
    PAGE/gts/pagecontent/2019-07-15 http://schema.primaresearch.org/
    PAGE/gts/pagecontent/2019-07-15/pagecontent.xsd">
<Creator></Creator>
<Created>2022-03-14T11:42:27</Created>
<LastChange>2022-03-14T11:42:27</LastChange></Metadata>
<Page imageFilename="P-2151.jpg" imageWidth="2524" imageHeight="
    4096">
<TextRegion id="tempReg35756468456854457908 9">
```

for f in *.png ; do convert "$f" "../alljpg/${f%.*}.jpg" ; done.

[2]More information available at https://www.primaresearch.org/tools/Aletheia

```
<Coords points="0,0 1,0 1,1 0,1"/>
<TextLine id="l4">
<Coords points="264,2496 264,2810 2301,2810 2301,2496"/>
<Word id="w10">
<Coords points="264,2496 264,2810 2301,2810 2301,2496"/>
<TextEquiv>
<Unicode>GENERATION</Unicode></TextEquiv></Word>
<TextEquiv>
<Unicode>GENERATION</Unicode></TextEquiv></TextLine>
...
```

Figure 4.5

4.2 Implementation

In this section I briefly described the structure of the experiment scripts and the function used inside them.

I wrote all testing scripts and auxiliary functions in Python programming language. The scripts were written as Jupyter Notebook environment and saved in the corresponding file format `.ipynb`, lately these notebooks were saved as `.py` Python scripts. The set of utility functions is located in a file called `utils.py`. All experiments were run within Google Colaboratory[3] which provides a 12 GB RAM and a GPU Nvidia K80 with 12 GB memory.

In the experiments I have chosen three methods to be tested, namely, Tesseract, EasyOCR and keras-ocr. Each of them has corresponding Python package. For Tesseract I tried two approaches, one with pure Tesseract and another with CRAFT detection tool followed by Tesseract recognizer.

4.2.1 Script Description

There are four scripts for the tested methods. Each script begins with downloading required packages, that are not already installed in Google Colab environment. This is followed by importing packages to the environment. Then Google Drive is mounted, because that is where image datasets are stored and where is located a file with utility functions. After that the utility functions can be imported, too.

The next stage is dataset loading. Each dataset is loading separately because tests are performed gradually for each dataset within each method. Images and

[3]Google Colaboratory (Colab for short) is an product from Google Research. It allow Google users to write and execute python code within a web browser using a remote computer and its computing power. In the free version of Colab the computational resources are limited and vary over time due to the demands of other users. The product is available at `https://colab.research.google.com/`.

respective ground truths are loaded and saved in variables. The list of image and text data are sorted so that corresponding data have the same indices.

Another step is the prediction. During this phase models are loaded and detection and recognition parameters specified.

Then metrics are computed. First the predictions are arranged into a same format. Then intersection over union metric for bounding boxes is computed and character error rate for comparing predicted and original texts. Averages of these metrics throughout an image are made for each image and these results are saved. Optionally one can save an image with the original image and bounding boxes and annotations drawn over it.

The pipeline can be summarized in a set of steps.

1. Installation of packages that are not in the default Google Colab setting.
2. Importing dependencies for the project including custom utility functions from `utils.py`.
3. Google Drive of the user that runs Google Colab is mounted. (Authentication is needed)
4. Image files and ground truth files are loaded. Ground truth is converted from files to tuple of string and integer coordinates.
5. If desired a short preprocessing of images is done (grayscale, Otsu thresholding).
6. Setup of parameters for OCR.
7. Model loading.
8. Prediction.
9. Conversion of prediction to a unified format (List of tuples of string and integer coordinates for each image, then joined to a list, which contains prediction for all images.)
10. Computation of evaluation metrics IoU an CER.
11. Saving and visualizing results.

4.2.2 Function Description

In the rest of this chapter the description of testing Python scripts is provided as well as comment on functions used in testing scripts. Note that int he following function definitions respective docstrings are omitted as they provide similar information as the descriptions, but in a briefer version.

Functions for Extracting Ground Truth

I created the following functions in order to extract ground truth information from text files – in most cases XML files. Each dataset unfortunately uses a different form of annotations. The styles differ in order of data, tag names, even in type of the recorded data (sometimes individual characters and their position is written or information about curvature). For XML parsing I utilized the The ElementTree XML API, which is imported as xml.etree.ElementTree. Information in XML files is stored in a tree where individual values can be accessed gradually using tags and their attributes.

All functions return annotations in the following format : `(label, [[top left X, top left Y], [bottom right X, bottom right Y]])`, a tuple of label and bounding rectangle coordinates in an array. If the images in the dataset were scaled then coordinates in annotations need to be scaled with same ratio. This ratio can be passed as a function argument and is default to one.

Function 4.1: read_gt_ctw_test

```
def read_gt_ctw_test(data, scaling_ratio=1):
    # one line = one bounding polygon : list of coordinates, each
    separated by commas, last is the text inside
    # there are #### before each text, two additional ## no text
    recognized

    annotations = []
    with open(data, "r") as file:
        for line in file:
            line = line.rstrip('\n')
            text = line.split("####")
            label = text[-1]
            coordinates = text[0].split(",")[:-1]
            c = [int(i) for i in coordinates]
            minX = min(c[::2])*scaling_ratio
            maxX = max(c[::2])*scaling_ratio
            minY = min(c[1::2])*scaling_ratio
            maxY = max(c[1::2])*scaling_ratio

            bbox_coords = np.array( [[minX, minY], [maxX, maxY]] )
            annotations.append((label, bbox_coords))

    return annotations
```

The function `read_gt_ctw_test` reads TXT files that contains the annotations for testing part of the SCUT-CTW1500 dataset.

Function 4.2: read_gt_ctw_train

```
def read_gt_ctw_train(xml_file, scaling_ratio=1):
    gt = []

    tree = ET.parse(xml_file)
    root = tree.getroot()

```

```
    # get values in this order: height, left coordinate, top coordinate, width
    for i, bbox in enumerate(root[0].findall('box')):
        # create list of integers with bounding box values, sort by attribute name
        # in case in different document there is a different order of attributes
        bbox_integer = [int(val) for key, val in sorted(bbox.attrib.items(), key = lambda el: el[0])]

        # calculate bottom coordinate of bounding rectangle x+width, y+height
        x_right= int((bbox_integer[1] + bbox_integer[3]) * scaling_ratio)
        y_bottom = int((bbox_integer[2] + bbox_integer[0]) * scaling_ratio)
        x_left = int(bbox_integer[1] * scaling_ratio)
        y_top = int(bbox_integer[2] * scaling_ratio)

        bbox_coords = np.array([[x_left, y_top], [x_right, y_bottom]])

        # get label
        label = root[0][i].find('label').text

        # create list of labels and corresponding bounding boxes
        gt.append((label, bbox_coords))

    return gt
```

The function `read_gt_ctw_train` reads the annotations from XML files for training part of the SCUT-CTW1500 dataset.

Function 4.3: read_gt_kaist

```
def read_gt_kaist(xml_file, scaling_ratio=1):
    gt = []

    tree = ET.parse(xml_file)
    root = tree.getroot()

    # some files has images as root tag,
    # some image as root tag (one tree layer (tag images) is missing)
    try:
        image_tag = root[0][2]
    except IndexError:
        image_tag = root[2]

    # get values in this order: height, width, x (left) coordinate, y (top) coordinate
    for i, bbox in enumerate(image_tag.findall('word')):
        # create list of integers with bounding box values, sort by attribute name
        # in case in different document there is a different order of attributes
```

```
18          bbox_integer = [int(val) for key, val in sorted(bbox.attrib
    .items(), key = lambda el: el[0])]
19
20          # calculate bottom coordinate of bounding rectangle x+width
    , y+height
21          x_right= int((bbox_integer[2] + bbox_integer[1]) *
    scaling_ratio)
22          y_bottom = int((bbox_integer[3] + bbox_integer[0]) *
    scaling_ratio)
23          x_left = int(bbox_integer[2] * scaling_ratio)
24          y_top = int(bbox_integer[3] * scaling_ratio)
25
26          bbox_coords = np.array([[x_left, y_top], [x_right, y_bottom
    ]])
27
28          # get label
29          label = ""
30          for char in image_tag[i].findall('character'):
31              ch = char.get('char')
32              label += ch
33          # create list of labels and corresponding boundin boxes
34          gt.append((label, bbox_coords))
35
36      return gt
```

The function `read_gt_kaist` extract annotations from XML files with labels for KAIST Scene Text dataset.

Function 4.4: read_gt_bd

```
1  def read_gt_bd(data, scaling_ratio=1):
2      annotations = []
3
4      with open(data, "r", encoding='utf-8-sig') as file:
5          for line in file:
6              line = line.rstrip('\n')
7              text = line.split(",")
8              bbox_coords = np.array(
9                  [[int(text[0])*scaling_ratio, int(text[1])*
    scaling_ratio],
10                 [int(text[4])*scaling_ratio, int(text[5])*
    scaling_ratio]])
11             annotations.append((text[-1], bbox_coords))
12
13     return annotations
```

Annotations for the Born-Digital dataset are stored in simple TXT file, where there are only four coordinates of the four corners of each bounding rectangle and a word within the rectangele. These values are processed by the function `read_gt_bd`.

Function 4.5: read_gt_wien

```
1  def read_gt_wien(xml_file, scaling_ratio=1, resized_previously=
    False):
2      # poster dataset
3      # returns labels in a tuple - first contains coordinates (8
    numbers), second word (string)
```

```
4
5       tree = ET.parse(xml_file)
6       root = tree.getroot()
7       image_width = root.find('{http://schema.primaresearch.org/PAGE/
        gts/pagecontent/2019-07-15}Page').get('imageWidth')
8
9       if resized_previously:
10          scaling_ratio = scaling_ratio / int(image_width)
11
12      root.iter('{http://schema.primaresearch.org/PAGE/gts/
        pagecontent/2019-07-15}TextRegion')
13
14      annotations = []
15      for word in root.iter('{http://schema.primaresearch.org/PAGE/
        gts/pagecontent/2019-07-15}Word'):
16          coordinates = word.find('{http://schema.primaresearch.org/
        PAGE/gts/pagecontent/2019-07-15}Coords').get('points')
17          text = word.find('{http://schema.primaresearch.org/PAGE/gts
        /pagecontent/2019-07-15}TextEquiv')
18          # extract only top left and bottom right coordinates and
        apply scale
19          coords = coordinates.split(" ")
20          topL = coords[0].split(",")
21          bottomR = coords[2].split(",")
22          bbox_coords = np.array(
23                      [[int(int(topL[0])*scaling_ratio), int(int(topL
        [1])*scaling_ratio)],
24                      [int(int(bottomR[0])*scaling_ratio), int(int(
        bottomR[1])*scaling_ratio)]])
25          annotations.append((text[0].text, bbox_coords))
26
27      return annotations
```

The function `read_gt_wien` takes the label and coordinate values form a XML files for the Vienna City Poster Dataset. The XML files produced by the software Aletheia are in a format created by PRiMA Research. When going through the XML tree an information about the research organization is embodied in the tag names.

Functions for Editing Images

Function 4.6: image_text_crop

```
1  def image_text_crop(images, filenames, ground_truth, one_file=True,
       result_folder='./results', skip_longer_than=40):
2      # test if there are not more gts than images
3      # else the for loop will never get to those exceeding image
       count
4      gt_length = len(ground_truth)
5      if len(images) > gt_length:
6          images = images[:gt_length]
7
8      if not os.path.isdir(result_folder):
9          os.mkdir(result_folder)
```

```
10
11      all_texts = []
12      for i, img in tqdm(enumerate(images)):
13          name, ext = os.path.splitext(filenames[i])
14
15          # count regions in one image - used for file naming
    purposes
16          region = 1
17
18          for text, bbox in ground_truth[i]:
19              if text is None:
20                  continue
21              if len(text) > skip_longer_than:
22                  continue
23              # select image within coordinates (bbox)
24              cropped = img[bbox[0,1]:bbox[1,1], bbox[0,0]:bbox[1,0]]
25
26              # create image file:
27              # name in format "original-00region.ext"
28              new_name = name + '-' + str(region).zfill(3)
29
30              if np.size(cropped):
31                  cv.imwrite(os.path.join(result_folder, new_name +
    ext), cropped)
32                  # create  text annotation file(s)
33                  if one_file:
34                      all_texts.append(new_name + ext + '\t' + text)
35                  else:
36                      # one file for each image with word
37                      all_texts.append(new_name + ext + '\t' + text)
38                      with open(os.path.join(result_folder, new_name
    + '.gt.txt'), 'w') as f:
39                          f.write(text)
40                  region += 1
41
42      if one_file:
43          with open(os.path.join(result_folder, 'gt.txt'), 'w') as f:
44              for line in all_texts:
45                  f.writelines(line+'\n')
46
47      return all_texts
```

The function `image_text_crop` crops and saves images based on bounding boxes provided in ground truth for each text region in an image. This is done for all images in `images` list. Also a text file is created with a corresponding text annotation. The name of the resulted file bears the original image name and a number is attached for each text region. The boolean parameter `one_file` allows to save ground truths only to a single file. Each line consists then of the name of the cropped image and its annotation. The parameter `skip_longer_than` skips ground truths that have more characters than stated, because some OCR tools (such as keras-OCR) does not support long strings.

Function 4.7: shrink_all

```
1 def shrink_all(images, width):
```

```
    scaled = []

    for image in images:
        if image.shape[1] > width:
            ratio = width / image.shape[1]
            height = int(image.shape[0] * ratio)
            new_size = (width, height)
            scaled.append(cv.resize(image, new_size, interpolation=
    cv.INTER_AREA))
        else:
            scaled.append(image)
    return scaled
```

The function `shrink_all` returns a list of resized images to a given width. Images already smaller than width are kept unaffected. The resizing aspects ratio.

Function 4.8: bounding_rectangle

```
def bounding_rectangle(coordinates):
    x, y = zip(*coordinates)

    return np.array([[int(min(x)), int(min(y))], [int(max(x)), int(
    max(y))]])
```

The function `bounding_rectangle` Returns top left and bottom right coordinates of a rectangle, that is circumscribed to a polygon defined by coordinates. These are obtained from predictions for each text region.

Functions for Computing Metrics

Function 4.9: iou

```
def iou(pred_box, gt_box):
    # find intersection rectangle coordinates
    x_left = max(pred_box[0][0], gt_box[0][0])
    x_right = min(pred_box[1][0], gt_box[1][0])
    y_top = max(pred_box[0][1], gt_box[0][1])
    y_bottom = min(pred_box[1][1], gt_box[1][1])

    if x_right < x_left or y_bottom < y_top:
        return 0

    # compute intersection area
    intersection = (x_right - x_left) * (y_bottom - y_top)

    # compute union area
    pred_area = (pred_box[1][0] - pred_box[0][0]) * (pred_box[1][1]
    - pred_box[0][1])
    gt_area = (gt_box[1][0] - gt_box[0][0]) * (gt_box[1][1] -
    gt_box[0][1])
    union  = pred_area + gt_area - intersection

    # return iou
    return  intersection/union
```

The function `iou` computes intersection over union of two bounding boxes, as defined in 1.4. Both input parameters have to contain top left and bottom right coordinates of a bounding box rectangle.

Function 4.10: iou_image

```
def iou_image(pred_boxes, gt_boxes):
    ious = []

    # have to determine which prediction bounding box contains same
    (similar)
    #  text region as ground truth bounding box
    # find and save the best iou for a prediction box and gt box
    for pred_ind, pred in enumerate(pred_boxes):
        max_iou = 0
        max_ind = 0
        for gt_ind, gt in enumerate(gt_boxes):
            iou_value = iou(pred, gt)
            if (iou_value > max_iou):
                max_iou = iou_value
                max_ind = gt_ind

        # match words from prediction and ground thruth (indices)
        ious.append((max_iou, pred_ind, max_ind))

    return ious
```

The function `iou_image` computes intersection over union for all text regions in one image. Each parameter shall contain a list of two coordinates - (top left, bottom right) of a bounding box rectangle. Returns a list of tuples - each tuple consists of the highest iou value (thus having the biggest overlap), the index of predicted bounding box and the index of ground truth bounding box. The indices are taken from the given lists of bounding boxes or ground truths belonging to one particular image.

Function 4.11: group_text

```
def group_text(lst):
    grouped = []
    key = lambda x: x[2]

    for k, g in itertools.groupby(sorted(lst, key=key), key):
        list_data = list(zip(*g))
        grouped.append((sum(list_data[0]), list_data[1], k))

    return grouped
```

The function `group_text` is a utility function for matching corresponding strings in a list. The list that is passed as an argument is a list of tuples of IoUs returned from the function `iou_image`. First for each ground truth bounding box, which is determined by the ground truth index (in this case the third element of the tuple) and it is used as a key. The returned tuple consists of a sum of IoUs, a list of indices of predicted bounding boxes and ground truth index.

Function 4.12: compare_text_cer

```
def compare_text_cer(text, special_characters=False, case_sensitive=False, spaces=True, split=True):
    text_gt, text_pred = text
    # remove special characters and case sensitivity if necessary
    if not spaces:
        text_gt = "".join(char for char in text_gt if (char.isalnum()))
        text_pred = "".join(char for char in text_pred if (char.isalnum()))
    elif not special_characters:
        text_gt = "".join(char for char in text_gt if (char.isalnum() or char.isspace()))
        text_pred = "".join(char for char in text_pred if (char.isalnum() or char.isspace()))
    if not case_sensitive:
        text_gt = text_gt.lower()
        text_pred = text_pred.lower()

    corresponding_words = []

    if split:
        words_gt = text_gt.split(" ")
        words_pred = text_pred.split(" ")

        # list of words that are corresponding (based on levenshtein distance)
        # and cer value. (=tuple of three elements)
        # for every predicted word find its corresponding gt wordle
        for word_pred in words_pred:
            min_dist = (1000, (0, 0))
            min_gt_word = ""
            for word_gt in words_gt:
                l_dist = levenshtein_distance(word_gt, word_pred)
                if l_dist[0] < min_dist[0]:
                    min_dist = l_dist
                    min_gt_word = word_gt
            # count normalized cer (the result will be from 0 to 1), 1 is the worst
            # for computation we devide Levenshtein dist. by sum
            # of the length of the word and count of insertions performed
            if len(min_gt_word) > 0 and len(word_pred) > 0:
                cer = min_dist[0] / (len(min_gt_word) + min_dist[1][2])
            else:
                cer = 1
            corresponding_words.append((min_gt_word, word_pred, cer))

        # no split of words
    else:
        l_dist = levenshtein_distance(text_gt, text_pred)

        if len(text_gt) > 0 and len(text_pred) > 0:
            cer = l_dist[0] / (len(text_gt) + l_dist[1][2])
        else:
```

```
47                 cer = 1
48             corresponding_words.append((text_gt, text_pred, cer))
49
50     return sorted(corresponding_words)
```

The function `compare_text_cer` returns a sorted list of corresponding words. Each pair of corresponding words is represented as a tuple of ground truth string, predicted string and a CER value. The first parameter have to be a tuple of a single ground truth string and a predicted string. The argument `special_characters` ignores all characters except alphanumeric characters and space when `False`. This is applied to both predicted and ground truth strings. When the argument `case_sensitive` is `False` then all texts are set to lowercase. The `spaces` parameter allows only alphanumeric characters and all spaces are removed. The last one of arguments is `split` and it is used optionally depending on the OCR engine and on dataset. When `True`, strings from ground truth and prediction are split with space used as a separator. Then the Levenshtein distance is computed for each pair of ground truth and predicted word. The best one is chosen for each predicted word and returned together with the two closest words. In case of no split, the Levenshtein distance is computed directly for the strings in the original tuple `text`. It is recommended to use split when the OCR model tends to detect words that belongs to each other as separate words and ground truth marks them as a text line with multiple words. Or when ground truth is one-word and model predicts strings with multiple words together.

The function `levenshtein_distance` computes the Levenshtein distance (definition can be found in subsection 1.4). The implementation is taken from a console application called xer. The code is available from `https://github.com/jpuigcerver/xer/blob/master/xer`. It returns a tuple of counts of the three performed operations – substitution, deletion, insertion.

Other Functions

Function 4.13: correct

```
1  def correct(string, char_mistakes):
2      """
3      Find defined mistake in a string and replace it with defined
       correction.
4      Returns corrected string.
5      """correct
6
7      for oldchar, newchar in char_mistakes:
8          string = string.replace(oldchar, newchar)
9
10     return string
```

Function 4.14: correct_german

```
1  def correct_german(text):
2      # For all texts correct these mistakes
3      text = correct(text, [('iu','in'),('LN','IN')])
```

```
    length = len(text)

    # ! shall be at the end of the text, if not it is probably the
    letter I
    if length > 1 and text.find('!') != length-1:
        text = text.replace('!','I')

    # For longer texts other mistakes can be corrected
    if length > 2:

    char_mistakes = [('{','f'),('$','S'),('[','L'),('/','I'),('|','
    I'),('ETN','EIN'),
                     ('CE','GE'),('GH','CH'),('GU','QU'),('gu','qu')
    ,('IZ','TZ'),
                     ('UNO','UND'),('uno','und'),('Wlen','Wien'),('
    wlen','wien')]
    text = correct(text, char_mistakes)

    # No Y at the begining of german words
    if text.find('Y')==0:
        text = 'V' + text[1:]


    # Characters that do appear in text sometimes, but usually
    alongside numbers
    # not if there are no numbers
    char_difficult = [('1','I'),('{euro sign}','E')]
    for old, new in char_difficult:
        if text.find(old) is not -1 and text.replace(' ','').
    replace(old,'').isalpha():
            text = text.replace(old,new)

    if text.find('(') is not -1 and text.find(')') is -1:
        text = text.replace('(','C')

    spaces = text.count(' ')
    if  spaces > length/2 - 2:
        text = text.replace(' ','')

    return text
```

The function returns corrected text based on typical German language syllables and common mistakes caused by optical character recognizers.

Functions Used in Testing Scripts

For each testing method there is a function called `get_predicted_{method_name}` where the format that is returned by a particular method is converted to a specific format. This format is a tuple of `(text, [upper left coordinates, bottom right coordinates])`. It is the same format as is used for ground truth.

Two following functions manipulate with the IoU and CER scores.

Function 4.15: get_iou_cer

```
def get_iou_cer(ground_truth, predicted, special_chars=False,
    case_sensitivity=False, split=False):
     iou_images = []
     cer_images = []
     n_imgs = len(predicted)

     # loop through images:
     for i in range(n_imgs):
         # separate list on columns (iterate through tuples in the
    list)
         if len(predicted[i]) and len(ground_truth[i]):
             predicted_cols = list(zip(*predicted[i]))
         else:
             iou_images.append(None)
             cer_images.append(None)
             continue
         ground_truth_cols = list(zip(*ground_truth[i]))

         # take only coordinate arrays from list for each images
         pred_boxes = predicted_cols[1]
         gt_boxes = ground_truth_cols[1]
         iou_from_image = iou_image(pred_boxes, gt_boxes)
         iou_text_regions = group_text(iou_from_image)

         # take only labels for each image
         pred_labels = predicted_cols[0]
         gt_labels = ground_truth_cols[0]

         # compare corresponding labels
         # comparison is a list of all text regions on one image
         comparision = []

         for observation in iou_text_regions:
             gt_ind = observation[-1]
             pred_ind = observation[1]
             predicted_text = " ".join([pred_labels[i] for i in
    pred_ind])

             if split:
                 gt_text = " ".join([i for i in gt_labels if i is
    not None])
             else:
                 gt_text = gt_labels[gt_ind]

             if gt_text is None or predicted_text is None:
                 continue

             gt_pred_text = (gt_text, predicted_text)

             # comparision for one text region (on one image)
             comparision.append((compare_text_cer(gt_pred_text,
    special_characters=special_chars, case_sensitive=
    case_sensitivity, split=split)))

         iou_images.append((iou_text_regions))
```

```
        cer_images.append((comparision))

    return iou_images, cer_images
```

The function `get_iou_cer` returns IoU and CER metric for bounding boxes for all images for each text region detected in an image. Thus there are two tuples with the length of number of images in a dataset.

Function 4.16: get_iou_cer_average

```
def get_iou_cer_average(iou_images, cer_images):
    iou_in_image = []
    cer_in_image = []
    n_imgs = len(iou_images)

    for i in range(n_imgs):
        # calculate mean based on results
        if isinstance(cer_images[i], list):
            length = len(cer_images[i])
            mean_in_regions = average([average(list(zip(*cer_images[i][j]))[2]) for j in range(length) ])
            iou_in_image.append(average(list(zip(*iou_images[i]))[0]))
        else:
            mean_in_regions = None
            iou_in_image.append(None)

        cer_in_image.append(mean_in_regions)

    return iou_in_image, cer_in_image
```

The function `get_iou_cer_average` returns two tuples of average IoU and CER value for each image in dataset.

Functions for Final Visualizations

Function 4.17: get_results

```
def get_results(filenames, iou_in_image, cer_in_image):
    df_results = pd.DataFrame(list(zip(filenames, iou_in_image, cer_in_image)), columns =['Filename', 'IoU', 'CER'])
    mean_iou = round(df_results['IoU'].mean() * 100, 1)
    mean_cer = round((1 - df_results['CER'].mean()) * 100, 1)

    print(f"mean IoU accuracy = {mean_iou}%, mean CER accuracy = {mean_cer}%")
    display(df_results)

    return mean_iou, mean_cer, df_results
```

The function `get_results` prints returns average IoU and CER for the whole dataset and a dataframe with these metrics for each image.

Function 4.18: plot_results

```
def plot_results(image, ground_truth, predicted, size=15):
    # Create figure and axes
    figure, ax = plt.subplots(figsize=(size, size))

    # Display the image
    ax.imshow(cv.cvtColor(image, cv.COLOR_BGR2RGB), cmap=plt.
    get_cmap('gray'))
    ax.axis('off')

    for label, bbox  in ground_truth:
        topleft = bbox[0]
        height = bbox[1,1] - bbox[0,1]
        width = bbox[1,0] - bbox[0,0]

        # create and add rectangle
        rect = patches.Rectangle((topleft), width, height,
    linewidth=1, edgecolor='g', facecolor='none')
        ax.add_patch(rect)

        # add labels
        ax.text(topleft[0]+width, topleft[1],label,
    verticalalignment='top', color='g',fontsize=13, bbox=dict(
    facecolor='g', alpha=0.2, edgecolor='g'))

    for label, bbox  in predicted:
        topleft = bbox[0]
        height = bbox[1,1] - bbox[0,1]
        width = bbox[1,0] - bbox[0,0]

        # create and add rectangle
        rect = patches.Rectangle((topleft), width, height,
    linewidth=1, edgecolor='r', facecolor='none')
        ax.add_patch(rect)

        # add labels
        ax.text(topleft[0]-2, topleft[1]-5,label,verticalalignment=
    'top', color='r',fontsize=13, bbox=dict(facecolor='r', alpha
    =0.2, edgecolor='r'))

    # smaller white borders
    plt.subplots_adjust(left=0, bottom=0.1, right=1, top=0.9,
    wspace=0, hspace=0)

    return plt
```

The function `plot_results` returns a plot with an image and both predicted and ground truth bounding boxes and corresponding labels. The ground truths are marked with green color and predictions with red. Objects are drawn using the package matplotlib.

Chapter 5

Results

In this chapter achieved results are presented. I tested each method on four datasets as described in chapter 3 and 4. Next I will shortly described possible setups of each of the three methods I used in the experiments. It will be followed by discussion of the results of test divided into sections by the four datasets. I have divided the experiments by the four datasets and made comparisons. Prediction methods were examined with a different setting of parameters where possible.

EasyOCR

I used the version 1.5.0 of the EasyOCR python package. The experiments were run using a GPU. I used two different settings for EasyOCR engine – the basic one which tends to detect and recognize text as single word objects and the second setting merges close words together creating text lines of words that belong together.

Keras-OCR

The keras_ocr-0.9.1 package for Python was used in experiments. It was necessary to compute the predictions on GPU, due to the limited GPU on Google Colab and the size of the images needed to be processed one by one although Keras allows to process images in batches. Keras-OCR predictions are case insensitive, because the default model does not support uppercase letters. I trained the Keras recognition model on SCUT-CTW1500 dataset (on the one thousand training images), first a trained the default model with lower case alphabet, then with the same alphabet and a space character and finally with uppercase letters and a set of special characters. In the results discussion it will be shown that after training the model does not offer better results than the original model. THe original model is already very successful and was well trained on much larger datasets.

Each of the training took about an hour on Google Colab GPU. I set 100 epochs and an early stopping based on the value of validation loss. The validation data were taken from the train data as 20% of the whole training part. First the I used

early stopping after 10 non decreasing values, then after 30, but in both cases the model could not get any better than 14% validation loss.

Tesseract

The version of Tesseract installed on Google Colab is tesseract 4.0.0-beta.1 with leptonica-1.75.3, which means that the LSTM network is available. For testing I used the OCR Engine mode (OEM) 3, that is Legacy + LSTM engines. As for Page segmentation modes (PSM) I used numbers 3, 4, 6, 8 and 11, which are "Fully automatic page segmentation, but no OSD. (Default)", "Assume a single column of text of variable sizes.", "Assume a single uniform block of text.", "Treat the image as a single word." and "Sparse text. Find as much text as possible in no particular order." respectively. I used PSM 3 only initially and dropped it because prediction was strongly inefficient, so it is not included in result statistics. PSM 8 was used for testing Tesseract with CRAFT, because CRAFT crops an image into segments where there is only one word (or a short text line), therefore Tesseract needed to treat the image as a single word. The rest of mentioned PSMs were applied for examining the behavior of pure Tesseract engine.

Experiments

In this section the results of experiments are divided into four parts based on the four datasets used for testing.

In table 5.1 is explanation of terms used in experiments and result discussion.

Term	Explanation
split	If there is a split option, it denotes that predicted strings were chopped into individual words separated by spaces.
no split	It means that predicted string was not changed.
tuning	Parameters of the model, other than the default ones, were set to produce better predictions.
psm	Page segmentation mode – a parameter for Tesseract.

Table 5.1: Explanation of terms used in experiments.

Born-Digital

First, the Born-Digital Images dataset was tested with images in predictions and ground truth were set to be in lowercase and all special characters except for space were removed. There were 13 different testing runs with three tested methods. In Figure 5.1 are the results of this testing and in Table 5.2 is corresponding information about each run.

The best CER value was achieved by tuned EasyOCR engine, where no split was done on predicted text (letter D in Fig. 5.1). We can see from the graph and table, that EasyOCR and keras-OCR had significantly better results than Tesseract. Also it can be said that tuning of EasyOCR model results in higher both IoU and CER of approximately 20% more than with basic EasyOCR setup of parameters. Keras-OCR performs similarly with split and no split option and in both cases gives satisfactory results. The best IoU score was equal to 64.6% (letters G and H) and belongs to Keras-OCR model. Tuned EasyOCR can get up to 60.3% without tuning 20% lower than that.

When we compare Tesseract with its own detection model and Tesseract with CRAFT tool, we can see that CRAFT increases the CER metric by more than 12%, however the IoU metric fluctuates for all cases around 50%. Whether the image is in RGB color scheme or in grayscale has only a little impact on the results, generally it differed only by about 2%. Same minor difference is when split or no split option is set. If the Otsu thresholding was performed and the image was then binarized, results were worse than when Tesseract itself performs the binarization. Better results were when PSM Tesseract parameter was set to 11 rather then PSM 4.

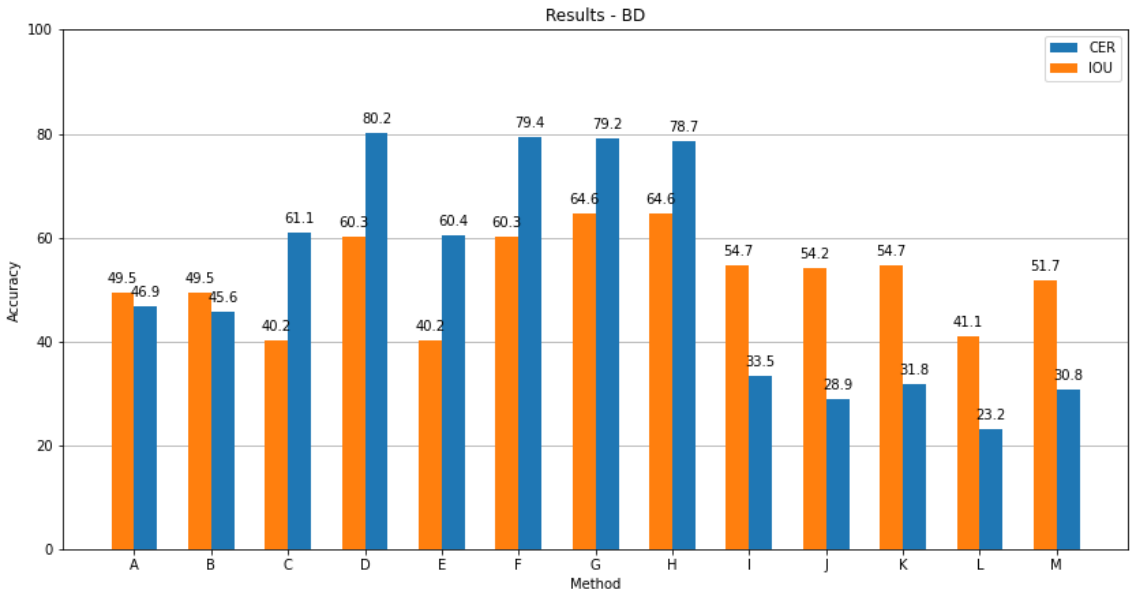


Figure 5.1: Results of experiments performed on Born-Digital Images dataset. Information about each method is in Table 5.2.

Next I performed tests with case sensitive option and with special characters

Key	Method	Properties
A	tesseract+CRAFT	no split, psm8
B	tesseract+CRAFT	split, psm8
C	easyOCR	no split, no tuning
D	easyOCR	no split, tuning
E	easyOCR	split, no tuning
F	easyOCR	split, tuning
G	keras-OCR	no split
H	keras-OCR	split
I	tesseract	colored, no split, psm11
J	tesseract	binary, split, psm11
K	tesseract	colored, split, psm11
L	tesseract	binary, split, psm4
M	tesseract	colored, split, psm4

Table 5.2: A table of keys for methods and parameters of experiments performed on Born-Digital Images dataset for Figure 5.1.

included, this option caused a decrease in CER value due to the fact that there are lots of special characters and uppercase words. However, it is good to have a recognizer that can predict more than lowercase strings even at the cost of slightly reducing the accuracy.

SCUT-CTW1500 dataset

The SCUT-CTW1500 dataset was again first tested with predictions and ground truth were set to be in lowercase and all special characters except for space were removed. 14 different testing were run. In Figure 5.2 are the results of the testing and the corresponding information about each run is in Table 5.3.

This time the best CER value (76.4%) was achieved by keras-OCR model, where split was performed on predicted text (letter H in Fig. 5.2). It can be seen from the graph and table, that EasyOCR and keras-OCR had again significantly better results than Tesseract. In testing of CTW1500 dataset this time keras-OCR generally performed better than EasyOCR in CER metric by roughly 10%. The tuning of EasyOCR model results in lower both IoU and CER of approximately 2% less than with basic EasyOCR setup of parameters. This might by probably due to the fact that EasyOCR model was trained on data similar to CTW1500 dataset rather then Born-Digital Images dataset Keras-OCR performs simlarly with split and no split option and in both cases gives satisfactory results.

The Comparison of Tesseract with its own detection model and Tesseract with CRAFT tool shows that with CRAFT the CER metric increased by almost than

20% as well as the IoU metric that increased by more than 30%. Plain Tesseract results are again the worst and are not heavily influenced by color scheme or slight scaling changes.

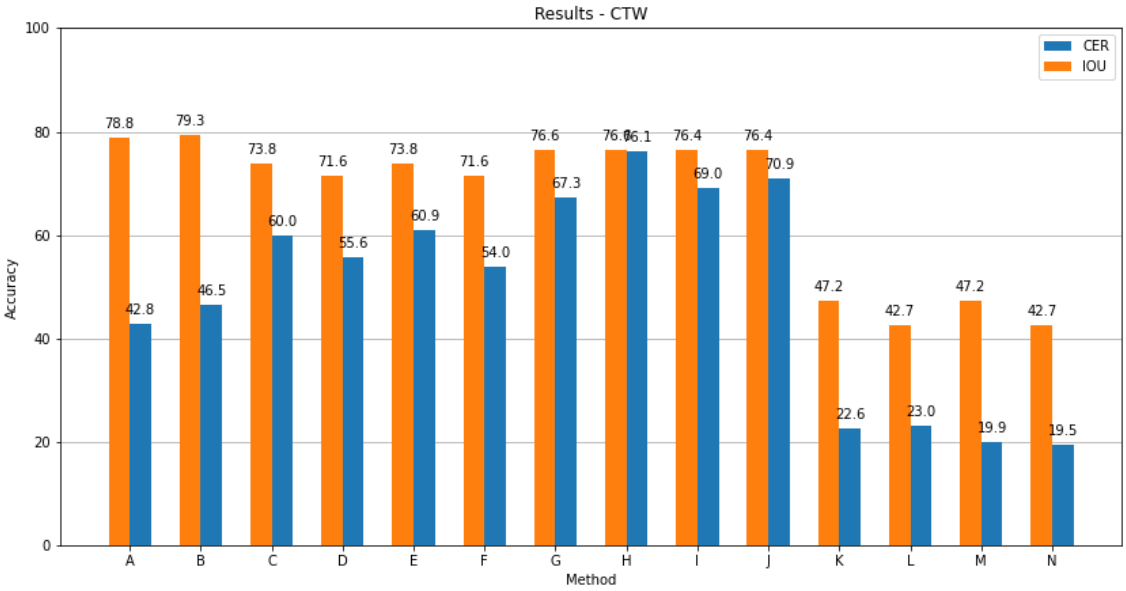


Figure 5.2: Results of experiments performed on CTW dataset. Information about each method is in Table 5.3.

Key	Method	Properties
A	tesseract+CRAFT	no split
B	tesseract+CRAFT	split
C	easyOCR	nosplit, notuning
D	easyOCR	no split, tuning
E	easyOCR	split, notuning
F	easyOCR	split, tuning
G	keras-OCR	original image width, no split
H	keras-OCR	original image width, split
I	keras-OCR	original image width, split, trained spaces
J	keras-OCR	original image width, split, trained
K	tesseract	3000px image width, no split
L	tesseract	color, no split, psm11
M	tesseract	split, psm11
N	tesseract	color, split, psm11

Table 5.3: A table of keys for methods and parameters of experiments performed on SCUT-CTW1500 dataset for Figure 5.2.

KAIST Scene Text Database

Testing of performance of the three methods on KAIST dataset with case insensitivity and no special characters was done in 22 different runs. Eight of them were on KAIST bitmap images, as these images do not have a misleading background the results are significantly better than on original photographes. The results can be seen in Figure 5.3 and the corresponding information is in Table 5.4. The best results on the bitmap images part of KAIST dataset were performed by EasyOCR dataset with no splitting and CER value is 82.2% high (letter B in Fig. 5.3). This method with the same setup is also best with KAIST dataset unedited photographs. Due to the character of the dataset for all cases the no split option is distinctly better.

Keras-OCR had slightly lower values of CER than EasyOCR and IoU values are even lower by generally 15%. Still it is by 20% higher than results of plain Tesseract and comparable with Tesseract and CRAFT combination. Tesseract had this time best but still way too low results with PSM 6. PSM 4 led to CER value as low as 18%. Color scheme did not have a distinguishable impact on the results.

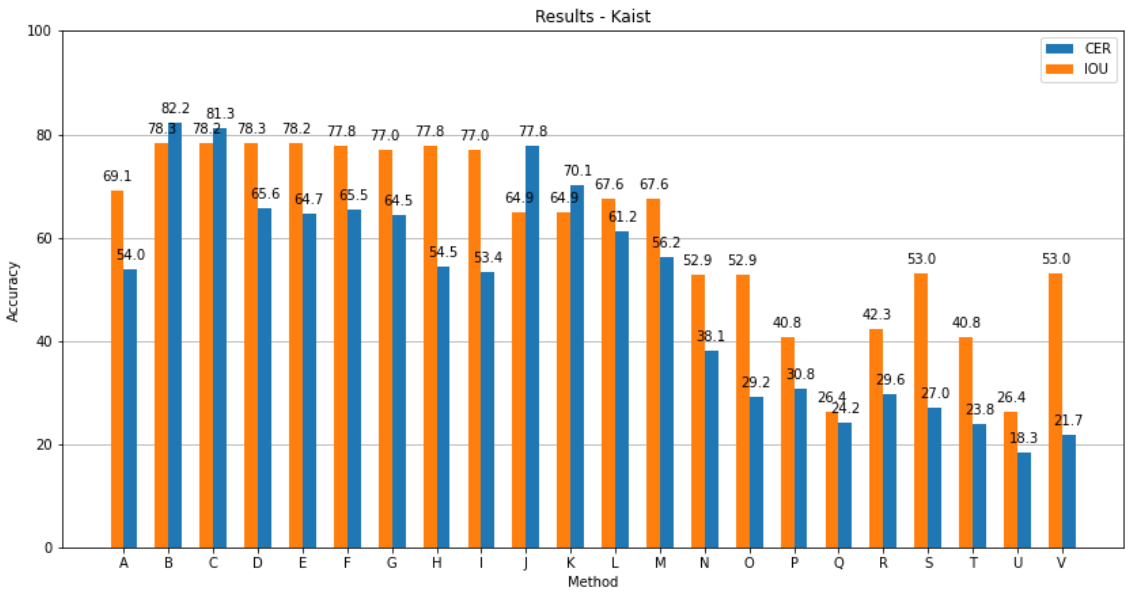


Figure 5.3: Results of experiments performed on KAIST Scene Text Database. Information about each method is in Table 5.4.

Key	Method	Properties
A	tesseract+CRAFT	no split
B	easyOCR	bmp, no split, no tuning
C	easyOCR	bmp, no split, tuning
D	easyOCR	bmp, split, no tuning
E	easyOCR	bmp, split, tuning
F	easyOCR	no split, no tuning
G	easyOCR	no split, tuning
H	easyOCR	split, no tuning
I	easyOCR	split, tuning
J	keras-OCR	bmp, no split
K	keras-OCR	bmp, split
L	keras-OCR	no split
M	keras-OCR	split
N	tesseract	bmp, no split, psm11
O	tesseract	bmp, split, psm11
P	tesseract	colored, no split, psm11
Q	tesseract	colored, no split, psm4
R	tesseract	colored, no split, psm6
S	tesseract	no split, psm11
T	tesseract	colored, split, psm11
U	tesseract	colored, split, psm4
V	tesseract	split, psm11

Table 5.4: A table of keys for methods and parameters of experiments performed on KAIST Scene Text Database for Figure 5.3.

Vienna City Poster Dataset

The Vienna City Poster Dataset was detected and recognized with the same method as previously mentioned datasets. Even though the dataset has only a little over 250 images, it is that large that the RAM size available in Google Colab struggled to work with the dataset as whole I had to split the dataset into two parts and perform computations on them separately. Then I combined the corresponding results. Specifically, the first group included first 123 images, the second the rest, where data were sorted by name of the files.

I used different parameters that can be set in each method. However, in the result table 5.5 and graph 5.4 I selected only the best ones within each method. Thus the results contains thirteen CER and IoU values. In the featured results for

all cases except one I tested only alphanumeric characters without the sensitivity to their case.

Due to the fact that this dataset is mainly in German language, in all models I set German as the language that is to be predicted. The defualt was English for all models. I implemented possible language correction which takes common syllables that appear in German. The function is described in 4.14. For some predictors these corrections were not necessary and predicted similarly without them.

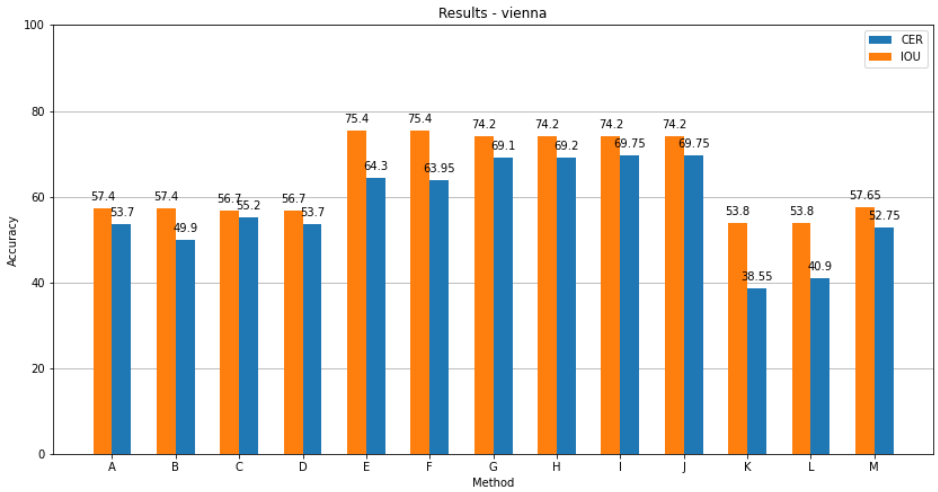


Figure 5.4: Results of experiments performed on Vienna City Poster Dataset. Information about each method is in Table 5.5.

The best results were obtained by EasyOCR and Keras-OCR. The highest IoU score is equal to 75.4% and it belongs to EasyOCR method with its parameters tuned to create tighter bounding boxes than pure EasyOCR (letters E and F in Fig. 5.4). In this case the CER values ranged around 64% and correction for German language was involved. Keras-OCR is responsible for the highest CER value – 69.75%, for both with and without correction. Though the IoU value is sligtly smaller than the best one – 74.2% (letters G-J).

As EasyOCR is able to predict also special characters and can be sensitive to case in its result I tested the two best settings (letters E and F) with this option. The CER scores lowered by approximately 3.5%. Which means that even with case sensitivity and more characters the model can give very good results. In contrast with Keras-OCR which cannot predict special characters and ignores text case without training. To gain this aibility, I trained the Keras-OCR model on the Vienna Dataset. I used the the second group of the dataset data as a training images and the first 123 images as testing ones. After seventeen minutes, which accords with twenty epochs, the training ended, because last ten epochs the validation loss stopped decreasing. Unfortunately the trainind dataset would need more data. After tests the IoU score was equal to 73.2% and CER score was 72.1% without my language corrections and with split applied to predictions. These numbers shows that Keras-OCR outperforms other models in both with simple characters and with special

Key	Method	Properties
A	Tesseract+CRAFT	split
B	Tesseract+CRAFT	case sensitive, includes special characters, split
C	easyOCR	no split, no tuning, correction
D	easyOCR	no split, no tuning, no correction
E	easyOCR	no split, tuning, correction
F	easyOCR	split, tuning, correction"
G	keras-OCR	no split, correction"
H	keras-OCR	no split, no correction"
I	keras-OCR	split, correction"
J	keras-OCR	split, no correction"
K	tesseract	no split, psm11"
L	tesseract	split, psm11"
M	tesseract	split, psm4"

Table 5.5: A table of keys for methods and parameters of experiments performed on Vienna City Poster dataset dataset for Figure 5.4.

characters.

Tesseract (letters K-M in Fig. 5.4) performance, similarly as with other tested datasets, was the worst. Usually the problem was caused by poor detection of the individual words. Very often objects were detected as letters, this happens when the PSM 11 was used. This eleventh option – find as much text as possible in no particular order – sounds ideal, because in the posters the text mainly does not keep an order and is present in various locations, orientations and sizes. CER values ranged around 40%. However, PSM 4, which assumes a single column of text of variable sizes, performed better with alike accuracy as Tesseract recognizer with CRAFT detector or as untuned EasyOCR, CER values reached neraly up to 53%. This comparision was unrealistic for scene text datasets.

When CRAFT package was used as a detector and cropped detected words were sent to Tesseract recognizing tool and treated as a single word (PSM 8). Then the CER score was equal to 53.7% (letters A and B in Fig. 5.4).

Result Conclusion

The previous section contains a discussion over achieved results. From this discussion it can be concluded that Keras-OCR and EasyOCR have the best results for all datasets. The CER score fluctuates between 65% and 80%. The upper boundary is very good and the lower also indicates acceptable predictions. The detection statistics, IoU, for these two winning methods is between 63% and 75%.

It can also be concluded that Tesseract gave overall the worst results. When performed on scene text image data, it returned CER value usually around 20%, on born-digital data results were around 30% and for the most important Vienna dataset it returned values ranging from 38% to nearly 53%. However these numbers are still lower than of any other method. When I changed the default detector in Tesseract to CRAFT detection tool, for all datasets the IoU statistics slightly increased and for the CER statistics a significant rise occured. In cases of SCUT-CTW1500 dataset and KAIST Scene Text Database the CER value doubled. A statement can be made regarding the page segmentation mode 11 of Tesseract tool – PSM 11, used for images where we want to find as much text as possible, causes that Tesseract finds much more words in the image and mistakes patterns for letters. This causes that a problem when interpreting the metrics. The IoU can falsely increase, because small false bounding boxes can cover parts of ground truth boxes bounding some large unrecognized word.

In 5 there are examples of images selected from the Vienna City Poster Dataset. Predictions of methods and ground truths are displayed on the images can be compared with each other. I also selected in the examples a couple of images with fonts difficult to recognize.

It is important to say that if there was a bigger labeled dataset for testing the methods, the score values would be more precise and the very difficult fonts and images would not have such an impact on the statistics as they did.

Conclusion

The aim of this paper was to examine different methods of optical character recognition (OCR) and test these methods on multiple datasets, including Vienna City Poster Dataset provided by the City Library of Vienna.

First, I introduced the topic optical character recognition. I defined text detection and text recognition and described individual approaches used in both of these task. Then I briefly mentioned types and examples of available datasets, that are related to text recognition tasks. I also described two main evaluation metrics that are applied to text predictions. I followed up on these information when implementing the testing of methods.

In the second chapter I expanded the widely used approach applied in OCR – neural networks. I gradually described different types of networks from the simplest ones to the more complex ones, that are used in OCR tools.

In the following chapter is a description od four OCR tools I had chosen for experiments – namely CRAFT, Tesseract, EasyOCR and keras-ocr. All of them are freely available and are accessible in Python language.

The last, most important, chapter contains a part of the implementation of testing scripts and a discussion of achieved results. In this chapter I also described the datasets I had selected for the experiments. I have chosen three free datasets and I created a fully labeled dataset of 257 images from given poster data with a manual labeling tool Aletheia.

I came up with a method of matching ground truth and predictions in image based on two evaluation metrics – Intersection over Union and Character Error Rate. By establishing the right match between original and predicted label I filtered most of the mistakes caused by wrong detection. For the German language, which is major in Vienna City Poster Dataset, I proposed possible swaps of characters in most common German syllables.

I have chosen Python as the language in which the testing experiments are written. Mainly to its popularity in machine learning tasks and also due to the fact that all selected OCR tools are supported. I performed over sixty tests. I tested the four different methods and I changed their parameters. For each dataset I provided a statistics with results of individual methods with different settings. I have found out that from the four free methods. EasyOCR and keras-OCR, both available as Python packages, give the best results no matter the given dataset. At the same time I came to a conclusion that Tesseract, a favourite OCR tool for scanned doc-

uments, is very unsuccessful when it comes to detection and recognition of text in images. I was able to obtain a 80% accuracy of characters in prediction compared to the ground truth with EasyOCR software.

The general accuracy of results was around 70% when the two most successful methods were used. This number would be more precise if the testing datasets were larger. However, this would need more time both for humans, which manually label the dataset and for computers to perform predictions.

Another possible improvement lies in further research and it is to extend the ability of a recognizer to work better with a contextual information. Because as humans we obtain this information by quickly scanning an image. For example when posters and advertisements are considered it is probable that a product name might occur multiple times without changes. This knowledge helps us to guess illegible words. Neural networks used in OCR however successful still need more of this ability to guess a result when struggling to give a precise prediction.

Bibliography

[1] D. Abueidda, S. Koric, R. Abu Al-Rub, C. Parrott, K. James, and N. Sobh. **A deep learning energy method for hyperelasticity and viscoelasticity**. *A deep learning energy method for hyperelasticity and viscoelasticity*, 01 2022.

[2] Jaided AI. **Jaided AI - Distribute the benefits of AI to the world**. Last accessed 20 May. 2022. [Online]. Available from: `https://www.jaided.ai/`.

[3] Jaided AI. **Jaidedai/EasyOCR**. Last accessed 13 Jun. 2022. [Online]. Available from: `https://github.com/JaidedAI/EasyOCR`.

[4] Y. Baek, B. Lee, D. Han, S. Yun, and H. Lee. **Character region awareness for text detection**. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9365–9374, 2019.

[5] R. C. Carrasco. **Text digitisation**. Last accessed 5 Jul. 2022. [Online]. Available from: `https://sites.google.com/site/textdigitisation/`.

[6] X. Chen, L. Jin, Y. Zhu, C. Luo, and T. Wang. **Text recognition in the wild: A survey**. *ACM Computing Surveys (CSUR)*, **54**(2):1–35, 2021.

[7] **COCO-Text V2.0**. Last accessed 22 Jun. 2022. [Online]. Available from: `https://bgshih.github.io/cocotext/`.

[8] Robust Reading Competition. **Overview - born-digital images (web and email)**. Last accessed 21 Jun. 2022. [Online]. Available from: `https://rrc.cvc.uab.es/?ch=1`.

[9] IBM Cloud Education. **What are recurrent neural networks?** Last accessed 4 Jul. 2022. [Online]. Available from: `https://www.ibm.com/cloud/learn/recurrent-neural-networks`.

[10] M. Gjoreski, G. Zajkovski, A. Bogatinov, G. Madjarov, D. Gjorgjevikj, and H. Gjoreski. **Optical character recognition applied on receipts printed in macedonian language**. 04 2014.

[11] R. C. Gonzalez and R. E. Woods. *Digital Image Processing (4th ed.)*. Pearson, 2018. ISBN 9353062985.

[12] R. Haldar and D. Mukhopadhyay. **Levenshtein Distance Technique in Dictionary Lookup Methods: An Improved Approach**, 2011. Available from: https://arxiv.org/abs/1101.1232.

[13] D. Hughes and N. Correll. **Distributed Machine Learning in Materials that Couple Sensing, Actuation, Computation and Communication**. 06 2016.

[14] **The IIIT 5K-word — Graviti**. Last accessed 22 Jun. 2022. [Online]. Available from: https://gas.graviti.com/dataset/graviti/IIIT5KWord.

[15] **Kaist scene text database**. Last accessed 10 Jun. 2022. [Online]. Available from: http://www.iapr-tc11.org/mediawiki/index.php/KAIST_Scene_Text_Database.

[16] **keras-ocr documentation**. Last accessed 29 Apr. 2022. [Online]. Available from: https://keras-ocr.readthedocs.io/en/latest/index.html.

[17] **keras-OCR**. Last accessed 29 Apr. 2022. [Online]. Available from: https://pypi.org/project/keras-ocr/.

[18] S. Kim, T. Hori, and S. Watanabe. **Joint CTC-Attention based End-to-End Speech Recognition using Multi-task Learning**. 09 2016.

[19] M. Liao, B. Shi, and X. Bai. **Textboxes++: A single-shot oriented scene text detector**. *IEEE transactions on image processing*, **27**(8):3676–3690, 2018.

[20] M. Liao, B. Shi, X. Bai, X. Wang, and W. Liu. **Textboxes: A fast text detector with a single deep neural network**. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[21] Jingchao Liu, Xuebo Liu, Jie Sheng, Ding Liang, Xin Li, and Qingjie Liu. **Pyramid mask text detector**. *arXiv preprint arXiv:1903.11800*, 2019.

[22] Y. Liu, L. Jin, S. Zhang, C. Luo, and S. Zhang. **Curved scene text detection via transverse and longitudinal sequence connection**. *Pattern Recognition*, **90**:337–345, 2019.

[23] **Text recognition data - Visual Geometry Group - University of Oxford**. Last accessed 21 Jul. 2022. [Online]. Available from: https://www.robots.ox.ac.uk/~vgg/data/text/#sec-synth.

[24] **Technology — The technology for converting books and documents into electronic files.** Last accessed 27 Jul. 2022. [Online]. Available from: https://www.abbyy.co.il/?categoryId=72050&itemId=168963.

[25] T. Phan, P. Shivakumara, S. Tian, and Ch. L. Tan. **Recognizing Text with Perspective Distortion in Natural Scenes**. pages 569–576, 12 2013.

[26] Z. Raisi, M. A Naiel, P. Fieguth, S. Wardell, and J. Zelek. **Text detection and recognition in the wild: A review**. *arXiv preprint arXiv:2006.04305*, 2020.

[27] Clova AI Research. **Clovaai/Craft-pytorch: Official implementation of character region awareness for text detection (CRAFT)**. Last accessed 29 Apr. 2022. [Online]. Available from: `https://github.com/clovaai/CRAFT-pytorch`.

[28] A. Rosebrock. **Intersection over union (IOU) for object detection**, Apr 2022. Last accessed 5 Jul. 2022. [Online]. Available from: `https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/`.

[29] H. Sak, A. W. Senior, and F. Beaufays. **Long short-term memory recurrent neural network architectures for large scale acoustic modeling**. In *INTERSPEECH*, pages 338–342, 2014.

[30] B. Shi, X. Bai, and C. Yao. **An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition**, 2015. Available from: `https://arxiv.org/abs/1507.05717`.

[31] R. Smith. **An overview of the Tesseract OCR engine**. In *Ninth international conference on document analysis and recognition (ICDAR 2007)*, **2**, pages 629–633. IEEE, 2007.

[32] **The street view text dataset**. Last accessed 22 Jul. 2022. [Online]. Available from: `http://vision.ucsd.edu/~kai/svt/`.

[33] **Synthtext in the wild dataset - Visual Geometry Group - University of Oxford**. Last accessed 21 Jul. 2022. [Online]. Available from: `https://www.robots.ox.ac.uk/~vgg/data/scenetext/`.

[34] Tesseract-Ocr. **Tesseract-OCR/tessdoc: Tesseract documentation**. Last accessed 6 May. 2022. [Online]. Available from: `https://github.com/tesseract-ocr/tessdoc`.

[35] M. Ul Hassan. **VGG16 - convolutional network for classification and detection**, Feb 2021. Last accessed 6 Jul. 2022. [Online]. Available from: `https://neurohive.io/en/popular-networks/vgg16/`.

[36] W. Wang, E. Xie, X. Li, W. Hou, T. Lu, G. Yu, and S. Shao. **Shape robust text detection with progressive scale expansion network**. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9336–9345, 2019.

[37] Yuliang-Liu. **Yuliang-Liu/curve-text-detector**. Last accessed 13 Jun. 2022. [Online]. Available from: `https://github.com/Yuliang-Liu/Curve-Text-Detector`.

[38] X. Zhou, C. Yao, H. Wen, Y. Wang, S. Zhou, W. He, and J. Liang. **East: an efficient and accurate scene text detector**. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 5551–5560, 2017.

[39] E. Zvornicanin. **Differences between bidirectional and Unidirectional LSTM**, Feb 2022. Last accessed 4 Jul. 2022. [Online]. Available from: `https://www.baeldung.com/cs/bidirectional-vs-unidirectional-lstm`.

Image Examples of Results

A selection of images from the Vienna City Poster Dataset is presented, with predictions and ground truth bounding boxes and labels. Red colour is for predictions and green for ground truth. Also I added comments on used method and achieved CER score. Some images are cropped because in the upper part of the picture is no text and results are then better comparable. First I have chosen images with good results and well legible text, then I have chosen images with complicated text for detection and recognition. They are readable by a human but machines might struggle because they remind more of images than letters, or are only half visible and strong contextual information is needed.

In table 5.1 is explanation of terms used in experiments and result discussion.

P-117050

This image has well readable letters and for all tested models this image was not difficult to read and only minor mistakes occurred.

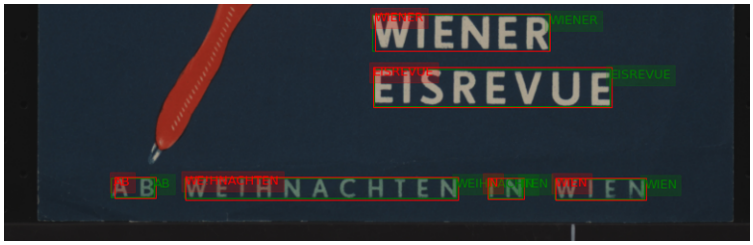


Figure 5: EasyOCR, tuning

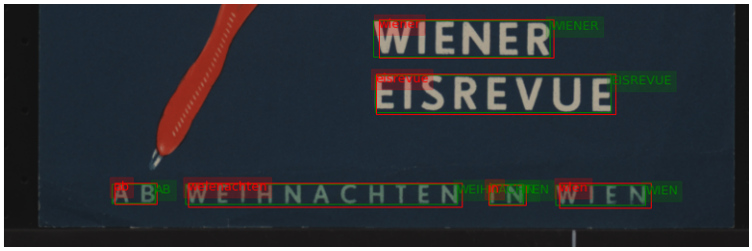


Figure 6: Keras-OCR, untrained

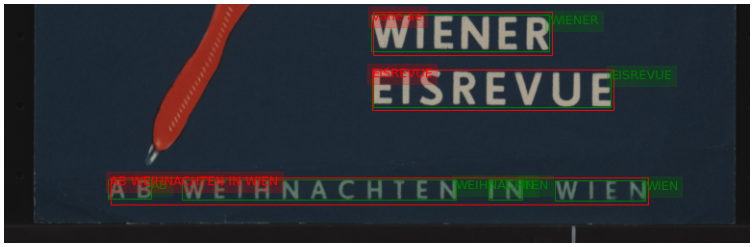
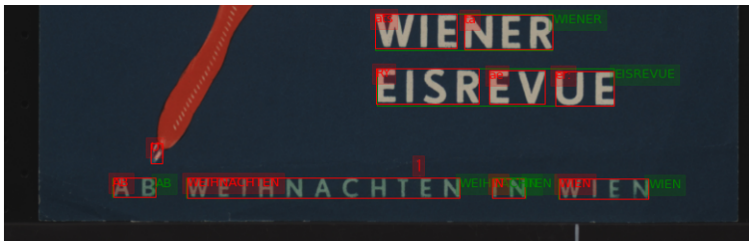


Figure 7: Tesseract+CRAFT

(a) PSM 11



(b) PSM 4

Figure 8: Tesseract

P-2638

(a) tuning

(b) no tuning

Figure 9: EasyOCR

P-315617

P-236873

This image has a text that is supposed to look like northern lights in a night sky and is not easy to read even by humans. None of the models were able to identify the wavy text.

P-229767

This image is a challenge for the detector and recognizer because there is a large number 10. The height and width of the number is same as the dimensions of the whole image and on top of that the number zero is not completely visible. Then there is curved text, which is also challenging.

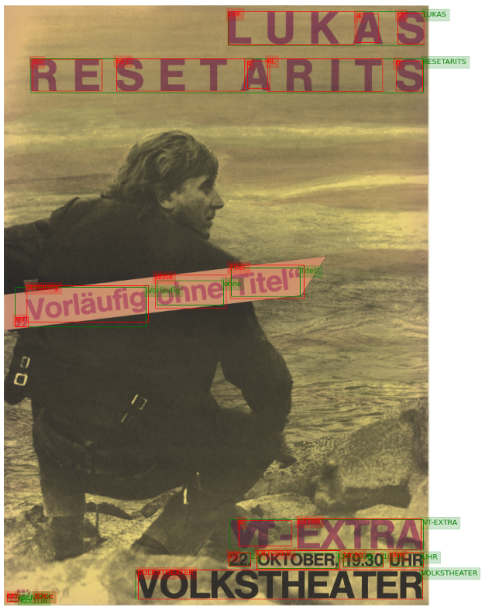
(a) Keras-OCR, trained

(b) Keras-OCR

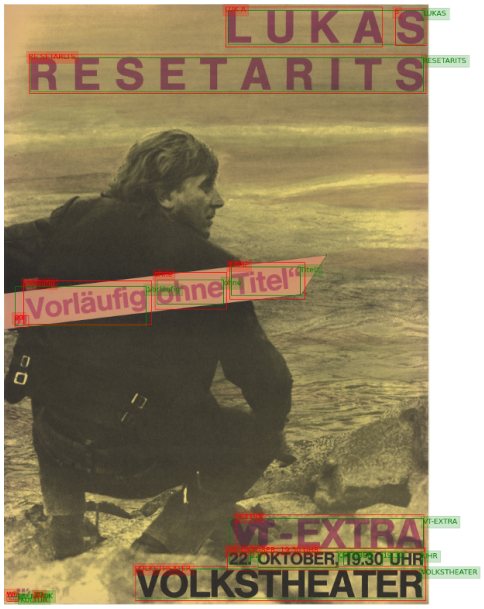
Figure 10: Keras-OCR

Figure 11: Tesseract+CRAFT

Figure 12: Tesseract, PSM 11. Tesseract found many non existing words and tried to predict them.



(a) tuning



(b) no tuning

Figure 13: EasyOCR

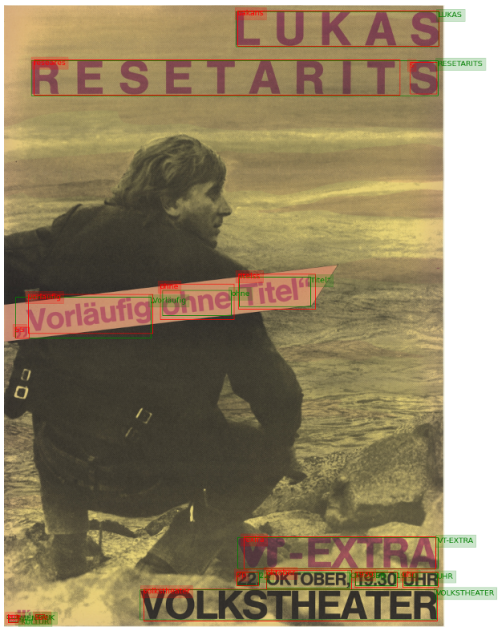


Figure 14: Keras-OCR, untrained

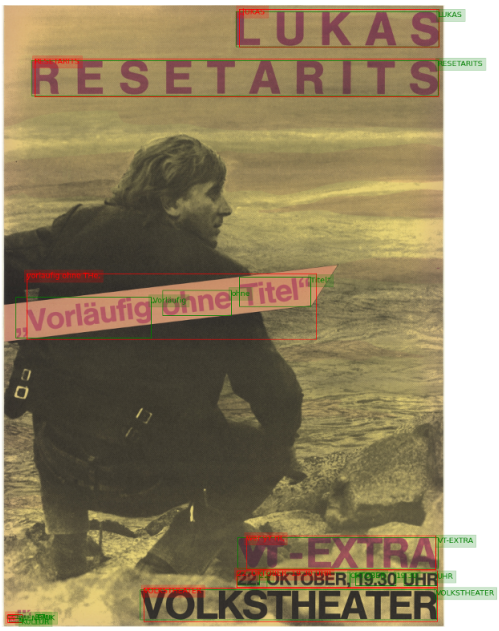
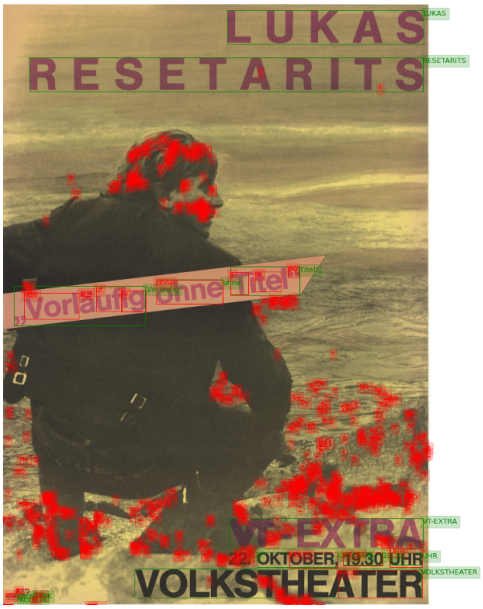
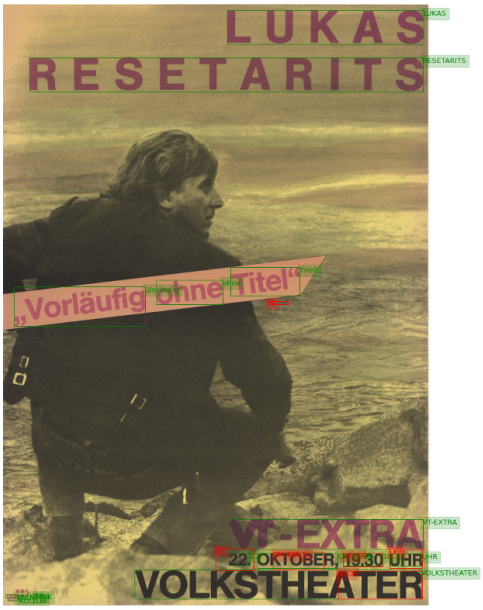


Figure 15: Tesseract+CRAFT

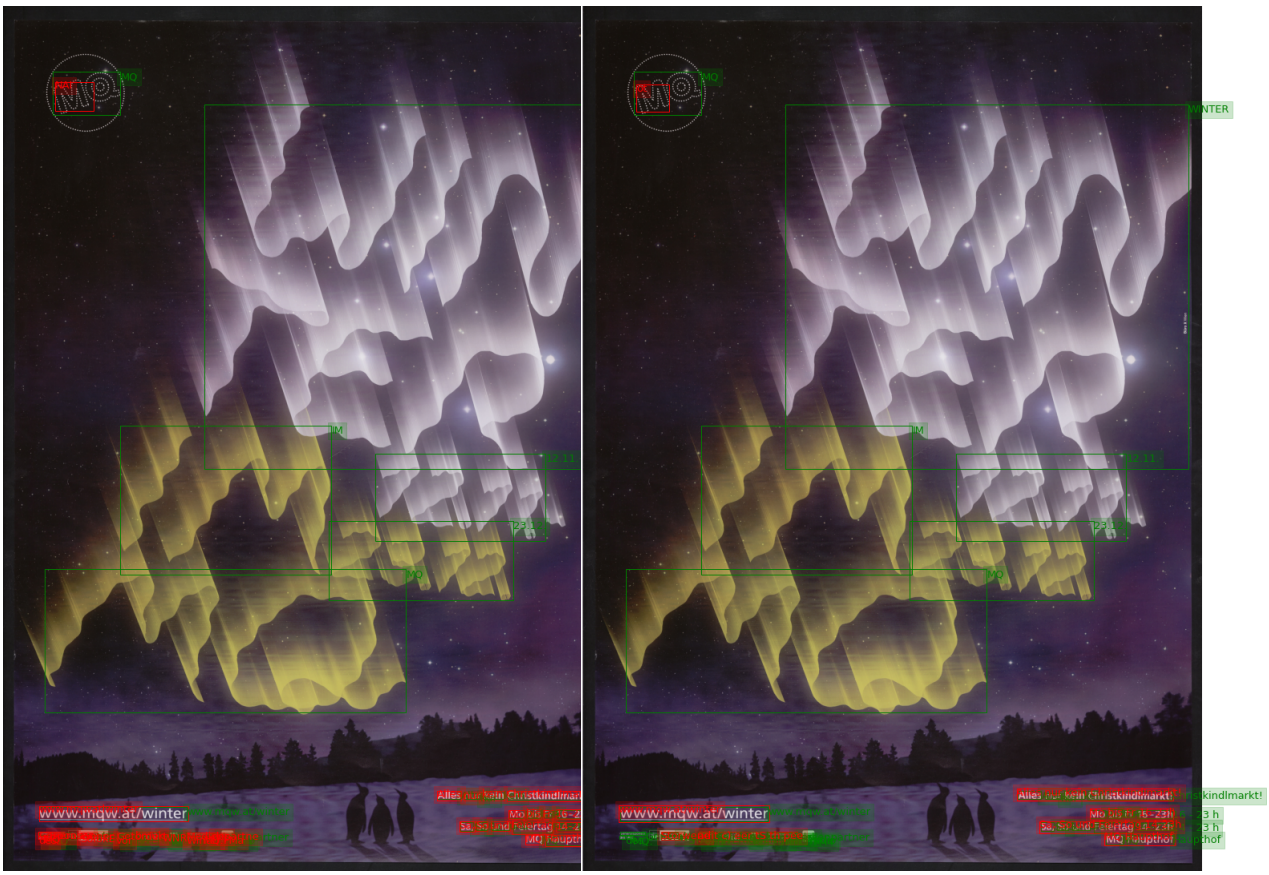


(a) PSM 11. Tesseract found many non existing words and tried to predict them.



(b) PSM 4

Figure 16: Tesseract



(a) EasyOCR (b) Tesseract+CRAFT

Figure 17: Tesseract

Figure 18: EasyOCR

P-310877

This image has a very specific font, which without former knowledge is very difficult to recognize.

Figure 19: EasyOCR on image P-

P-231248

This image demonstrates that vertical text is more complicated for the recognizer although it is quite legible.

Figure 20: EasyOCR on image P-