

**M A S A R Y K
U N I V E R S I T Y**

FACULTY OF INFORMATICS

**Detection of Czech Words in
pictures**

Bachelor's Thesis

ERIC VINCENT VALČÍK

Brno, Fall 2022

**M A S A R Y K
U N I V E R S I T Y**

FACULTY OF INFORMATICS

Detection of Czech Words in pictures

Bachelor's Thesis

ERIC VINCENT VALČÍK

Advisor: doc. Mgr. Radek Pelánek, Ph.D.

Department of Machine Learning and Data Processing

Brno, Fall 2022



Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Eric Vincent Valčík

Advisor: doc. Mgr. Radek Pelánek, Ph.D.

Acknowledgements

I want to thank my supervisor, doc. Mgr. Radek Pelánek, Ph.D., for guidance and valuable insights in the process of creating and finishing this thesis, and for granting me access to the computational server. Without it, I would still be running the first iteration of the model on my laptop.

Further, I want to thank my family for their immense support during my studies, and last but not least, I want to take my friends for motivating me to keep pushing.

Abstract

The detection of text inside images is an essential topic of machine learning. This thesis further extends OCR engines that implement text detection from images to implement models for finding only the Czech language inside of images. The resulting model is developed to help translate a learning portal *Umimeto.org* [1] to other languages.

Keywords

OCR, machine learning, image analysis, language detection, python, data analysis, PyTorch

Contents

1	Definition of the Problem	3
1.1	Downloading the Images	3
1.2	Detecting Text in the Images	3
1.3	Working with the Generated Text	4
1.4	Evaluation, Performance Measurement	5
1.4.1	False Positives vs. False Negatives	5
2	State of the Art	6
2.1	Proprietary OCR Tools	6
2.2	Tesseract OCR	6
2.3	EasyOCR	7
2.4	Benchmarking Different OCR Solutions	8
3	Implementation	10
3.1	Preprocessing	10
3.1.1	Adding Margin	10
3.1.2	Thresholding	11
3.1.3	Rotating Images	12
3.2	CVS Data to Images	12
3.3	OCR Engines	12
3.3.1	PIL	13
3.4	Language Detection	13
3.5	Dictionary Check	14
3.5.1	Exceptions to this Check	14
3.6	Length Limit on the Words	15
3.7	Outer Join	15
3.8	Parameters	15
3.8.1	OCR Model	15
3.8.2	Rotation	16
3.8.3	Adjusting the Minimal Word Length	16
3.8.4	English Corpus - Take Only N Most Frequent Words	16
3.8.5	OCR Engine Confidence Threshold	16
3.9	The Result	16
3.10	Used Libraries and Technologies	17

3.10.1	Pandas	17
3.10.2	Python Notebooks	17
4	Training of the Model	19
4.1	Training Set	19
4.2	The Training Process	20
5	Evaluation	21
5.1	Testing Data	21
5.2	Testing Techniques	21
5.3	Mathematics Dataset	22
5.3.1	Positive Class	22
5.3.2	Negative Class	24
5.4	Informatics Dataset	24
5.5	Positive Class	24
5.5.1	Negative Class	24
5.6	Interpreting the Results	25
6	Conclusion	27
	Bibliography	28
A	Running the final model	30
A.1	Setup	30
A.2	Running the final model	30
A.3	Notes	30
A.4	GPU Optimizations	31
B	List of Electronic Attachments	32

Introduction

This thesis is done in collaboration with *Umimeto.org* [1], a learning portal focused mainly on elementary, middle, or high school students. The portal focuses on many fields, such as Mathematics, Czech, English, and Informatics. Users can strengthen their knowledge of these fields in an interactive way by playing games or solving puzzles.

In this thesis, I am developing a tool that gets an image as an input and outputs Czech text found in the image, if there is any. This output is further used to classify images into two classes, *with Czech text* and *without Czech text*. The motivation behind developing a tool for detecting Czech text in images is to make translating the whole *Umimeto.org* portal into different languages more accessible. The key goals are to not flag images that contain any text but flag those images containing Czech text that should be translated. For example, the model should not flag an image containing only the word *START* as containing the Czech language.

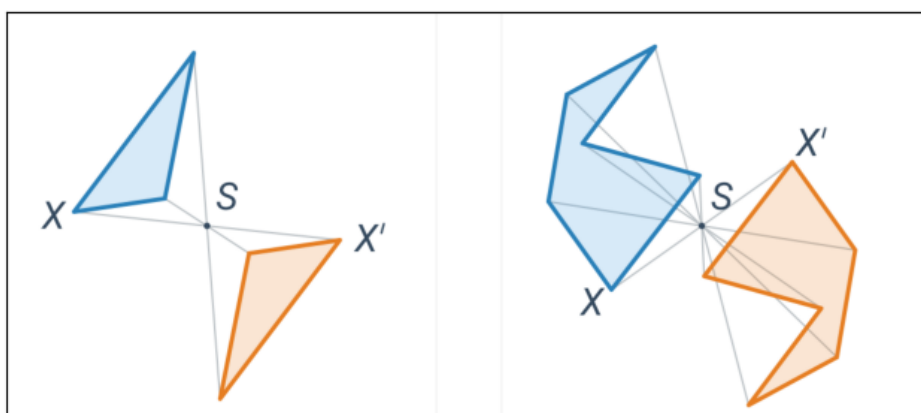
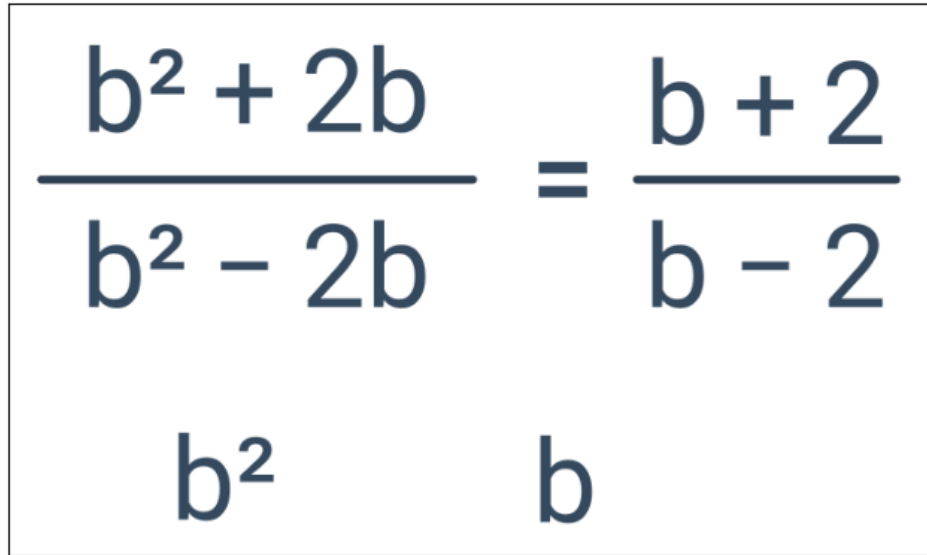


Figure 1: This picture contains text but should not be translated.

I solve the problem by running an OCR model on the input images, finding all texts in the image, and then further processing the generated text. The specific part of this project is postprocessing. The images can contain many universal words and characters that do not have to

be translated, so the main task is to differentiate between words that should be translated and that do not have to be.


$$\frac{b^2 + 2b}{b^2 - 2b} = \frac{b + 2}{b - 2}$$

b^2 b

Figure 2: Mathematical formulas do not have to be translated.

I am not focusing on implementing the OCR model. The main focus is applying available tools and connecting them to solve the problem. The focus is more on the specifics of the problem rather than on creating a general-purpose tool. This thesis results in a model that correctly (with some error percentage) flags images for translation with the help of an existing OCR model.

1 Definition of the Problem

The problem from the perspective of the *Uمیمeto.org* [1] team is simple. They provide a set of images they use on their website and expect a classification of each image in the set into two classes: *should be translated* (positive) or *does not have to be translated* (negative).

The text that should be translated is all Czech text, but text like mathematical formulas and notation or axis names x and y do not have to be translated. Also, English words that are recognized globally do not have to be translated as well. The boundary is not as clear for things like names. I decided to translate those names that are not often used in English, which are the majority of Czech names.

We can divide the problem into the following sections.

1.1 Downloading the Images

The first step is to take the CSV file and generate URLs on which the pictures are available. There are also multiple CSV files, one CSV file for each subject, for example, mathematics and informatics. The model has to respect this partitioning.

The CSV can have multiple types of assets: image, video, and audio. The path to the resource is not present in the record. Instead, there is a *directory* and *file name*. Putting these two together, we get a unique path for that resource within the website's file system.

The unique path of each resource does not yet work as a link that can be used to download the image. We must prepend the path with the correct URL to get a downloadable link.

1.2 Detecting Text in the Images

The model can now generate text for each image from downloaded images. This step is the most crucial; the data we get from the OCR model must be as accurate as possible.



Figure 1.1: Example of an image where text detection could be more difficult.

1.3 Working with the Generated Text

Here the model filters the generated text. Filter out the English words, and keep only the Czech words. The boundary between which words must be translated is not firmly given, so in this step, we experiment and try different configurations to get the best results.

After the filtering, we get the resulting words found in each image that our model flagged as *should be translated*. We can then generate a list of images that should be translated simply by adding images with some associated text. The images without any text are the ones we do not need to translate. A list of images flagged as *should be translated* along with the detected words is the result of the model.

1.4 Evaluation, Performance Measurement

The datasets have thousands of images each, so going through all of the images is not a viable option if we want to check the correctness of the classifications. In the following chapters, I detail how the training set is generated and how the model is trained. Also, the testing process is described in the Evaluation chapter.

1.4.1 False Positives vs. False Negatives

By the nature of the problem, the more important metric for us is the amount of *false negatives*. *False negatives* are those images that we classified as *does not have to be translated* but have Czech text that should be translated in them.

False negatives are a bigger problem than *false positives* because a translator will go through all the images that should be translated, so they can see if some of the images were wrongly classified and not translated them. Still, for the other class, they are classified as good to go in the other languages, so no one should go through them, and that means that an untranslated image could stay there undetected.

2 State of the Art

In this chapter, I explore the existing relevant work on OCR models and the detection of text inside images. I focus on the most advanced models and use cases. I am not focusing on research papers about OCR as the goal of this thesis was not to develop a better OCR model than the ones we have now but to find and use a solution with the existing OCR tools.

Valuable sources for navigating current OCR engines were Wikipedia [2] and AwesomeOCR [3]. Both sources offer a list of the most popular OCR engines, their license, portability to operating systems, the programming language they are written in, and the languages they support. The last update to the AwesomeOCR project was on Nov 2021, so it might not include the most recent information.

2.1 Proprietary OCR Tools

The proprietary tools provide an easy way of detecting and recognizing text inside images and pdfs. Google, for example, has its proprietary tools for OCR, such as Google Cloud Vision API [4]. Google also claims to detect and recognize handwritten text.

A benefit of using such tools is that the user gets state-of-the-art OCR without the need to configure the models, meaning they do not have to understand and set up the OCR models themselves. The disadvantage of using such a service is paying Google, usually per image or page, when OCR-ing pdfs.

Similar tools are provided by numerous big cloud providers, like Microsoft with Azure Computer Vision API [5] or Amazon with AWS Textract [6]. This thesis does not use proprietary tools; it only focuses on free and open-source solutions.

2.2 Tesseract OCR

Tesseract [7] is an open-source OCR project that dates back to the 1990s. It was a proprietary software owned and developed by Hewlett-Packard Laboratories. It was open-sourced in 2005.

The latest version, Tesseract 4, uses Long short-term memory (LSTM) neural networks and works by detecting lines of text, while the previous version, Tesseract 3, detects groups of characters. Either of these approaches can be used when using this tool.

This tool is written in C++ and has almost no external dependencies except an Input-Output (IO) library.

Tesseract OCR only provides the OCR engine as a C++ library and a Command Line Interface (CLI). However, third-party Graphical User Interfaces (GUIs) are available and linked on the Tesseract GitHub repository page.

Setting up Tesseract can be more problematic than using a cloud-based solution, but all the code is accessible and can be changed or improved. But Tesseract's precision is not as good in some specific cases.

Tesseract was my first option as the OCR engine, but the results from this model are not as good as those from EasyOCR [8]. The advantage is that the model works faster and does not require significant processing power. Still, I decided to use another model because the performance was not essential, and the outputs were for some images unacceptable.

2.3 EasyOCR

EasyOCR [8] is an open-source OCR tool using the *PyTorch* [9], an Artificial Intelligence (AI) library, and can be used out of the box with Python.

This model aims to be a framework with interchangeable parts for other OCR implementations, but it also has its OCR engine built into it. EasyOCR also provides the option to implement a custom OCR engine.

It can be used online without downloading, for example, on HuggingFace¹ webpage.

EasyOCR was my second and final option. The results were more precise, and all the wrongly detected text I found was only in *difficult cases* (for example Figure 1.1).

1. <https://huggingface.co/spaces/tomofi/EasyOCR>

The only drawback of using EasyOCR is its performance. EasyOCR is around 5-10 times slower than Tesseract on my machine. Because that would mean running the OCR on the whole dataset would take around 8 hours, I decided to use FI MUNI's Aura [10] server. Running EasyOCR on the whole database on this server took only around 30 minutes.

2.4 Benchmarking Different OCR Solutions

Benchmarking tools are developed to measure and compare the performance of the OCR tools. Dilmegani [11] describes one such tool and compares AWS Textract [6], Google Cloud Vision API [4], ABBYY FineReader 15, Microsoft Azure Computer Vision API [5], and Tesseract OCR Engine [7]. AWS Textract performed the best, with Google Cloud Vision being just as good or just behind.

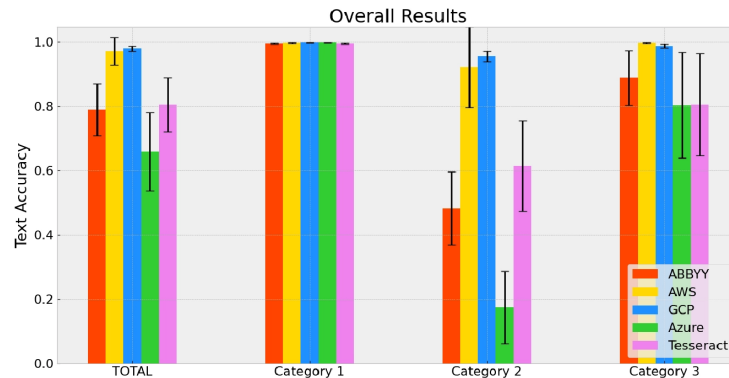


Figure 2.1: Chart from the cited article [11], showing precision for typed and handwritten text together.

AWS managed to get more than 98 % accuracy with typed and handwritten text, while, for example, Azure's solution has under 20 % accuracy for the handwritten category.

The only open-sourced project in the comparison, Tesseract, did not perform as well as the best proprietary solutions. Only Azure's OCR performed worse than Tesseract in some categories.

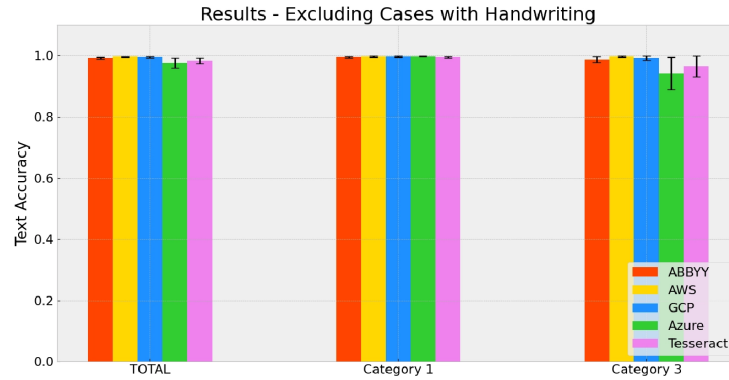


Figure 2.2: Chart from the cited article [11], showing precision only for the typed text.

Omitting the handwritten text, all OCRs performed exceptionally well, with minor differences.

In my experience, I had less success with Tesseract. On the other hand, Tesseract is around 8-10 times faster on my local machine. However, since I do not focus on the model's performance but on the precision of the results, I switch entirely to EasyOCR when generating the final results in this thesis because the results were more correct.

3 Implementation

I tried many configurations when implementing the final model and interchanged different pipeline parts. Here I try to explain most of the approaches I had when implementing all of the versions, not only the final one.

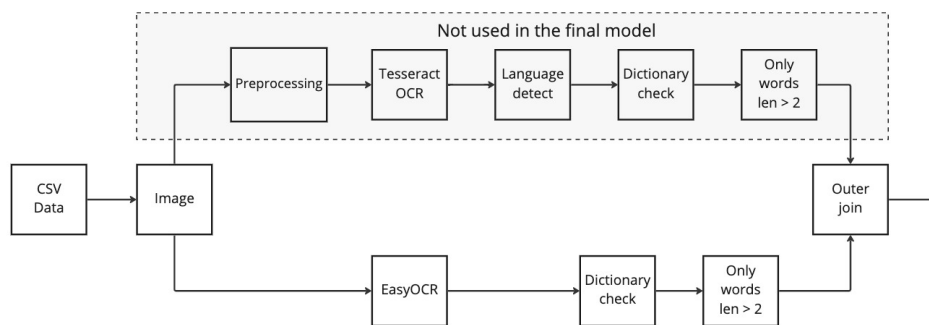


Figure 3.1: Diagram showing the final model along with the parts that ended up not being included in the final model.

3.1 Preprocessing

In this step, the model changes the images, so the detected text we generate with the OCR engine is more precise. The initial pipeline used multiple preprocessing techniques with varying results.

3.1.1 Adding Margin

The first technique added a margin to every picture before the detection.

This function takes an image as a parameter, adds a margin of a given color (or white if color is not specified), and returns the edited image. The result looks like the figure below.

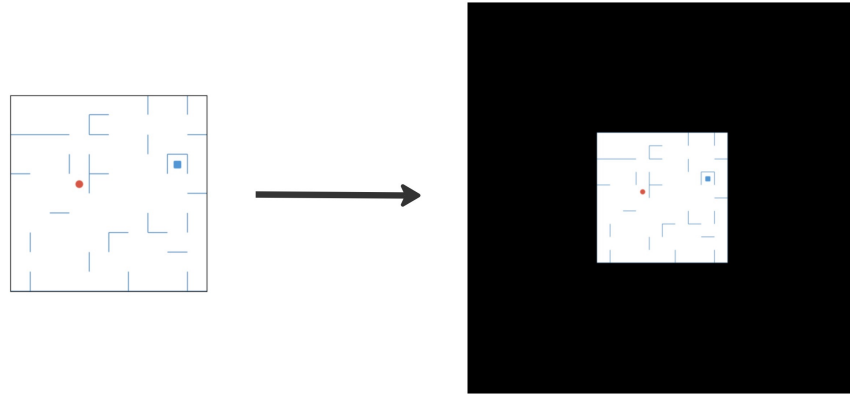


Figure 3.2: Image before and after applying the `add_margin` function with parameter `color='black'`.

This technique improved the model for specific cases, especially those where the text was located very close to the edges of the initial image.

3.1.2 Thresholding

Thresholding converts the image into a binary image in which every pixel is white or black. To determine which pixel is white or black, we use a threshold. If the pixel in the initial image is darker than the given threshold, this method sets the pixel to black and otherwise sets it to white.

This method is susceptible to the threshold. Finding a threshold that would work for every image is a complex problem and very hard to check. Also, we lose significant detail from each image by applying this preprocessing method.

This method only improved the model for specific cases but usually broke the pipeline completely for other cases.

3.1.3 Rotating Images

Sometimes the text inside the images is not perfectly horizontal. With this method, the model generates multiple images from a single initial one, groups them together, and sends them to the detection step.

In this step, the number of rotations can vary. For example, if we rotate the image by 45 degrees, we generate eight images from a single initial one to get all the possible rotations.

This step was helpful in some cases, and the model can never perform worse than without this preprocessing method, as the initial image is always included in the set of images sent to the next step.

All preprocessing methods are included in the Tesseract pipeline, but only the rotation preprocessing is present in the final model. The other methods are unnecessary or do not improve the outputs.

3.2 CVS Data to Images

As mentioned in the Definition of the Problem chapter, the input is a CSV file. We have to download all the images and save them locally. We have to download the images only once, so this step is not included in the pipeline of the model. The model itself works with the paths to images on the machine.

3.3 OCR Engines

With the images, the model has to find all text that is inside the images. This is the first step in the diagram where the model versions diverge. In the unused branch, the model used Tesseract for the OCR. In the current branch, the model uses EasyOCR for the OCR.

Even if the OCR engines are different, this step gets the same input and expects the same output for both engines. The input is an image, and the output is the text found in the image.

	5 dělí	9 dělí	8 dělí
6 dělí	2	8	2
2 dělí	5	1	4

Figure 3.3: Tesseract did not recognize any Czech text in this image.

As mentioned in the State of the Art chapter, EasyOCR performed in all observed cases as good as or better than Tesseract, and for some cases, Tesseract returned very unexpected results even for *relatively easy* images, as shown in the Figure above.

The Tesseract OCR expects an instance of PIL's Image class that should be detected, while EasyOCR expects a path to the image. Also, each engine's output structure is different and must be explicitly handled for the two engines.

3.3.1 PIL

PIL is a python library for working with images. It enables the user to open, save, edit or view images. PIL's object of an image was one of the types allowed as input for EasyOCR.

I also use PIL for viewing the images throughout my work. PIL and the library matplotlib provide an easy way to show loaded images directly in python.

3.4 Language Detection

Language detection is a step where the model tries to determine the language of the generated text. For this task, I decided to use a python library langdetect [12].

Langdetect is a python wrapper around Shuyo Nakatani's language detection library [13] implemented in Java. This library claims

it has over 99% precision, which is supported by several test results. Their language detection works exceptionally well with longer texts, but it worked well for this use case.

The main problem with this step is that images could have only one word in them, and that word could be shared between more than one language (usually Slavic languages). In such cases, the library can generate multiple languages, which are all correct but are not the Czech language.

After experimenting with this library, I decided not to include this step in the final model, as it created more trouble than good.

3.5 Dictionary Check

This step consists of two dictionary checks. The corpus I check is the Czech corpus syn2015 [14]. Every word not present in the Czech corpus is removed from the detected text.

With the Czech corpus, I hoped to remove all non-Czech words, but there are a lot of English words in the corpus. That's why I decided to introduce another dictionary check, with an English corpus containing one-third of a million most frequent English words on the Web [15]. The model does not use the whole corpus. It only uses the top 5000 words if we sort the corpus by frequency. The number of words can be changed with a parameter, for further tuning the model. Using the whole dataset didn't work as well as I expected, but the boundary can be changed for better results.

This check works oppositely. If a word in the detected text is also in the English corpus, I remove it from the detected text.

This step has the same purpose as the language check but, in practice, works better. This step is included in the final model.

3.5.1 Exceptions to this Check

This step is executed for most words, but the final model has an exception built in. The exception holds for every word that contains letters with diacritics used in the Czech language. That means if a

word contains any of letters *áčďěěňůřšťúůž*, the word automatically skips the Czech corpus check and is kept in the detected words.

3.6 Length Limit on the Words

The model removes all short words from the detected text in the last step. The best boundary turned out to be two characters, removing all words with two or fewer characters—this step helps remove false positives from the result. The boundary is set to 3 but can be changed with a parameter.

3.7 Outer Join

This step aims to have both pipelines in the final model (Tesseract pipeline and EasyOCR pipeline), unite their output, and deliver the results as the unification.

Again, in practice, this brought more trouble than good. The correct detections Tesseract’s pipeline made, the EasyOCR’s pipeline made as well. Still, Tesseract for some images also found text that was not in the image, bringing more false positives to the results.

3.8 Parameters

The model was iteratively improved in two ways in the training process. The first way is by introducing or removing parts of the pipeline, and the second is by adjusting the model’s hyper-parameters. I will list the hyper-parameters that can be adjusted for each model configuration.

3.8.1 OCR Model

The final pipeline has a parameter *engine*, where the user can specify the OCR model to be used in the pipeline. Allowed values for the current versions are **easyocr** and **tesseract**, but more could be added.

3.8.2 Rotation

The parameter *rotate* is a boolean parameter. If set to **true**, the OCR detection will run 4 times on the given input image, rotating the images by 90° after each detection, then concatenating the detected text and proceeding further in the pipeline. If it is set to **false**, the OCR detection is only run on the input image. **False** is the default value for this parameter.

3.8.3 Adjusting the Minimal Word Length

This parameter *min_word_len* is set to three by default. It determines how long a word should be, as discussed in the Length Limit on the Word section.

3.8.4 English Corpus - Take Only N Most Frequent Words

The parameter *n_english_words* introduces a limit on how many words from the English Corpus will be used in the English dictionary check. The corpus has one-third of a million words; without this parameter, many Czech words were deleted from the detected words. The default value of this parameter is 5000.

3.8.5 OCR Engine Confidence Threshold

When making detections of words in the image, the OCR engine also returns a confidence level for that word, ranging from 0 to 1. By setting the parameter *confidence_threshold*, we can adjust the minimal confidence in the detected word by the OCR engine for the pipeline to consider the detection. The default value for this parameter is 0, meaning all detections are used in the pipeline. This parameter is only supported when the used OCR engine is EasyOCR.

3.9 The Result

The last step is creating a CSV file with all the images classified as *should be translated*. The model generates this CSV by adding paths to all the images with their set of detected words being non-empty. Additionally, adds the detected text in the next column.

3.10 Used Libraries and Technologies

This section lists all the libraries and technologies I used throughout the model.

3.10.1 Pandas

Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. [16]. With pandas, users can arrange the data inside their DataFrames, similar to DataFrame from the programming language R. These DataFrames are virtual tables with each record stored as a row. Adding, removing, or altering the data within these DataFrames is similar to altering a table in a relational database.

Using pandas is crucial for this thesis, as working with data in python only with the built-in tools would add a lot more work and make the code more cluttered. Also, with pandas visualizing the data is more straightforward, and the visualized data is easier to read.

Figure 3.4: Example of a visualization of a DataFrame with image path and the OCR'd data

path	text
img/presouvani/presouvani-podminena-kouzla/presouvani-podminena-kouzla-13-clean.png	pokud oči
img/presouvani/presouvani-podminena-kouzla/presouvani-podminena-kouzla-6-clean.png	modrý klobouk motýlek pokud
img/presouvani/presouvani-podminena-kouzla/presouvani-podminena-kouzla-5-clean.png	klobouk klobouk pokud
img/presouvani/presouvani-podminena-kouzla/presouvani-podminena-kouzla-15-clean.png	uši oči ruce
img/presouvani/presouvani-podminena-kouzla/presouvani-podminena-kouzla-17-clean.png	oči pokud uši ruce nohy pokud
...	...
img/rules/vysvetleni-podminky-if.png	pokud obsahuje seno sen poslední sláma sláma písmeno
img/rules/vysvetleni-opakovani-zelva.png	opakuj dopředu doprava
img/rules/vysvetleni-opakovani-ovoce.png	opakuj opakuj opakuj opakuj
img/overview/up-prehled.png	stupeň stupeň želví grafika želva programování jednodušší úlohy dopomoci typická cílová skupina ...
metodika/informatika-letak.png	umíme informatiku všestranné procvičování informatiky pro revidované široké pokrytí témat pestré...

1188 rows x 1 columns

3.10.2 Python Notebooks

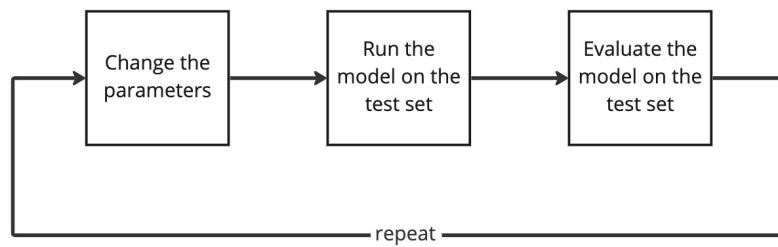
Python notebooks are a powerful tool for writing python code for scientific/data analytic purposes. The notebook remembers context (variables, functions) after completing each notebook segment, similar to a python shell window.

The python notebook is divided into cells: text cells, which introduce a more powerful way of documenting and commenting on the

code, and code cells, which are made of python code and can have their output. With libraries PIL and matplotlib, users can show images inside the notebooks.

4 Training of the Model

Figure 4.1: Diagram showing the iterative process of training the model.



The structure of the final model described in the chapter Implementation is partly a result of the training. Still, the chapter does not describe the training process and results of the model with different configurations.

This chapter will compare viable configurations of the model and evaluate their performance on a labeled training set.

4.1 Training Set

I used the dataset for Mathematics and Informatics to create a training set. I could not use the dataset as is since it is unlabeled. I created a subset of 1000 randomly selected images from the Mathematics dataset and 1000 randomly selected images from the Informatics dataset and labeled them manually. I then used this labeled dataset to evaluate the model after each modification.

I selected images equally from both datasets, so the model does not overfit one of the datasets.

4.2 The Training Process

The training process was iterative. One iteration consisted of modifying the model based on analyzing the wrongly classified images from the previous iteration and evaluating the iteration. By pre-labeling the training set, the evaluation process was fast, allowing us to compare the different model versions and generate new versions faster.

model	rotate	thres	eng_corpus_n	TP	TN	FP	FN
EasyOCR	true	0.5	25000	365	1612	10	13
EasyOCR	true	0.5	15000	365	1599	23	13
EasyOCR	true	0.5	10000	365	1594	28	13
EasyOCR	true	0.5	5000	365	1590	32	13
EasyOCR	true	0.0	10000	365	1574	48	13
EasyOCR	true	0.0	5000	365	1565	57	13
EasyOCR	false	0.5	5000	363	1600	22	15
EasyOCR	false	0.0	5000	362	1600	22	16
EasyOCR	true	0.62	5000	362	1597	25	16
EasyOCR	true	0.6	5000	362	1594	28	16
EasyOCR	true	0.5	35000	361	1616	6	17
EasyOCR	true	0.5	50000	360	1618	4	18
Tesseract	false	0.0	5000	338	1640	42	40

Table 4.1: Table of the measurements with different parameters, sorted primary by *false negatives* and secondary by *false positives*. TP stands for true positives, TN for true negatives, FP for false positives, and FN for false negatives.

In **Table 4.1**, we can see the individual iterations described by their hyper-parameter selection and evaluation. The process of choosing parameters was arbitrary. The suitable option would be to run a grid search on the parameters, but one iteration of the pipeline on the 2000 images runs around 20 minutes.

The best performing set of parameters for the model from the evaluated data is the first row in the table, and that is **rotate**=true, **threshold**=0.5, **n_english_words**=25000. These are the parameters used for the final model, evaluated in the next chapter, Evaluation.

5 Evaluation

In this chapter, I focus on the performance of the model. The model was trained on the training set, but to get an accurate estimation of how good the model is, we have to run it on data it was not trained on.

5.1 Testing Data

The model is tested by running it on the whole dataset, Mathematics, and Informatics, then picking 200 random images from each class classified by the model, 100 from the Mathematics dataset and 100 from the Informatics dataset. So, the evaluation is run on 400 images equally distributed among the classes and the two datasets.

5.2 Testing Techniques

As mentioned above, the more critical metric is the *false negatives*, rather than the *false positives*. The *false negatives* can be represented by the recall metric. Recall is defined in the following way.

$$\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

If the model minimizes the amount of *false negatives*, its recall will be close to 1.

But having recall as a single metric would not work, because, for example, classifying the whole dataset *positive* would yield perfect recall. That is why we have to introduce precision. Precision is a metric defined as follows.

$$\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

Recall is the more critical, but we have to find a balance where precision is reasonably good. Both values should be as close to one for a good model.

5.3 Mathematics Dataset

The dataset for mathematics has exactly 10 356 images. After running the model, the model flagged 861 images as *should be translated*, so the percentage of images flagged for translation is around 8.31 %.

5.3.1 Positive Class

Manually going through the randomly picked 100 images flagged as *should be translated*, I found 10 images that were wrongly classified, leaving us with an estimation of 10 % *false negatives* rate.

$7 < \sqrt{50}$	$\sqrt{20} < 5$	$8 > \sqrt{60}$	$\sqrt{30} < 6$
$9 = \sqrt{9^2}$	$\sqrt{6} > 2$	$7 = \sqrt{7^2}$	$\sqrt{10} > 3$
$4 >$		$5 <$	

Figure 5.1: These images were classified as having Czech text in them. The found word for the first image is *19ž* and for the second one are *ola* and *zla*.

The images above were classified for translation, although they contain no words. Then, because the word *19ž* contains a letter with Czech diacritics, the Dictionary check is skipped, and the word is accepted as a detected Czech word. And for the words *ola* and *zla*, they were considered Czech words by the model, which holds for the second word.

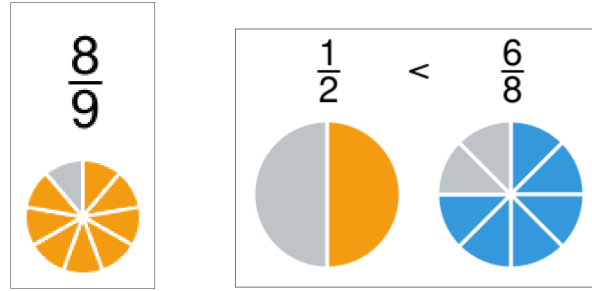


Figure 5.2: These images were classified as having Czech text in them. In both images, the detected word is *clo*.

In the figure above, we have another positive classification despite having no words in the image. The detections of the word *clo* by the OCR engine (EasyOCR) are made when the image is rotated to its side, so the numbers are rotated 90°. With some imagination, we can see where the model could see the word in the image. Nonetheless, this is another wrong classification of a relatively straightforward case.

Images with geometry notation were the biggest group of *false positives* for mathematics. For 7 of the 10 total *false positives*, the detected text was some combination of one-letter labels and symbols misinterpreted as letters. The model filters out most geometrical notation correctly, but some stay in the detected text, creating these false positive cases.

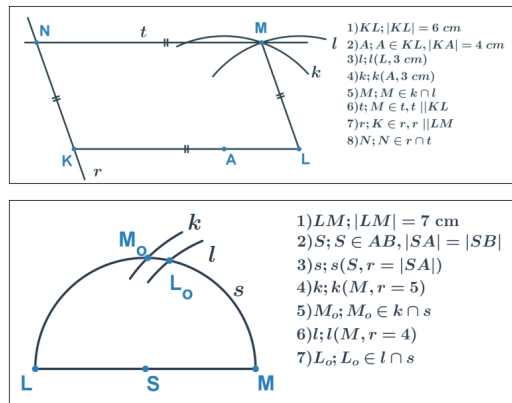


Figure 5.3: In the first image word *ikl* was detected. In the second words *ssr* and *ins*.

In the second image from **Figure 5.3**, the words *ssr* and *ins* are not in the image. Still, the OCR engine detected the words with a combination of mathematical symbols.

The estimated percentage of *false positives* can seem high. Still, because the model flagged as positive 861 images from the whole dataset, we can assume that around 86 of those images will be *false positives*. So we have estimated 86 wrong classifications for the whole dataset of size 10356.

5.3.2 Negative Class

In the randomly picked 100 images flagged as *does not have to be translated*, I found 0 images that were wrongly classified.

5.4 Informatics Dataset

The testing dataset is the dataset for Informatics. This dataset has 5 256 images in total. The model classified 1 798 images as *should be translated*, which is 34.2 % of the whole set.

5.5 Positive Class

Manually going through the randomly picked 100 images classified as *should be translated*, none of the images was classified wrongly.

5.5.1 Negative Class

In the randomly selected 100 images, I found only 1 image that was wrongly classified. The wrongly classified image is in the following figure.



Figure 5.4: This image was classified as having no Czech text in it.

If we look closely, under the title of the image, there are names, probably of the authors of the presentation. The names are written in gray and are not very readable so finding the text from this image is a relatively challenging task for the OCR engine.

We only found one image classified wrongly as negative, and the picture was relatively hard to classify.

5.6 Interpreting the Results

The estimated recall metric is .995, and the estimated precision is 0.95. Even if the precision could be higher, we reached almost perfect recall, our initial goal. Almost every image from the set *false positives* in the translation process will be found and will not remain undetected.

Let's consider the size of the two classes and try to estimate the percentage of wrongly classified images from our measurements. If there is 10 *false positives* per 100 for the Mathematics dataset and 0 for the Informatics dataset, then we have around 86 *false positives* for the whole dataset. We can estimate this by computing the formula

below for both datasets (Mathematics and Informatics) and adding the results together.

$$\frac{\text{found wrongly classified}}{100} * |\text{all classified}|$$

We can use the same formula for *false negatives*. We found *false negatives* only in the Informatics dataset, and there we found 1 wrong classification per 100 negative ones. Since there are exactly 3458 images classified as negative in the Informatics, if our false rate is 1 %, we get around 35 *false negatives* for the whole dataset. And since there were no *false negatives* found in the Mathematics dataset, we can estimate 35 *false negatives* for both datasets together.

We can further calculate the estimated percentages for each class, as shown in the table below.

	FP rate	FN rate	false classification rate
Mathematics	0.83 %	0 %	0.83 %
Informatics	0 %	0.66 %	0.66 %
both datasets	0.55 %	0.22 %	0.77 %

Table 5.1: Computed percentage of the number of errors among the datasets.

From the table, we can see that based on our estimation, less than 1 % of the images are wrongly classified, and *false negatives* form less than a third of these errors.

6 Conclusion

This thesis aimed to take an existing OCR Engine and, with its help, implement a model for the specific problem of detecting images with Czech text.

In chapter 1, we define the problem. The exact format of the input, we define the format of the output. Also, there are more investigated requirements for the model, for example, if *false positives* and *false negatives* have the same value when evaluating the model.

In the State of the Art chapter, we overview the existing OCR options, and I also add my own experience when working with the model, if there is any. There is also a comparison of the performance of the most used OCR Engines.

The chapter Implementation overviews the model, goes through all the pipeline steps, and explains the intuition behind each step. This chapter also partly describes the steps that were not included in the final model but were a part of the model at some point in development. It lists the model's parameters, but the model's training is described in the following chapter called Training of the Model.

The last chapter evaluates the final model on the provided datasets, estimates metrics about the model's performance, and goes through individual cases where the model classified the images wrong.

The resulting model classifies the images with an estimated average greater of 99 % accuracy and with most of the cases of wrong classification being on *false positives*, which are easier to be detected in practical use. To improve the model, we could introduce more parameters, for example, using only n most frequent words from the Czech corpus. Another improvement that, unfortunately, was too time-consuming to include in this thesis, is to run a grid search on more parameter configurations to find the perfect configuration.

Bibliography

1. *Umíme to - Online procvičování školního učiva* [online]. Umímeto team [visited on 2022-10-23]. Available from: <https://www.umimeto.org/>.
2. *Comparison of optical character recognition software* [online]. Wikipedia [visited on 2022-10-13]. Available from: https://en.wikipedia.org/wiki/Comparison_of_optical_character_recognition_software.
3. BAIERER, Konstantin. *Awesome OCR* [online]. [visited on 2022-10-13]. Available from: <https://github.com/kba/awesome-ocr>.
4. *Detect text in images* [online]. Google [visited on 2022-10-13]. Available from: <https://cloud.google.com/vision/docs/ocr>.
5. *Computer Vision* [online]. Microsoft [visited on 2022-10-13]. Available from: <https://azure.microsoft.com/en-us/products/cognitive-services/computer-vision>.
6. *Amazon Textract* [online]. Amazon [visited on 2022-10-13]. Available from: <https://aws.amazon.com/textract/>.
7. *Tesseract OCR* [online]. Tesseract maintainers [visited on 2022-10-13]. Available from: <https://github.com/tesseract-ocr/tesseract>.
8. *EasyOCR* [online]. JaidedAI [visited on 2022-10-13]. Available from: <https://github.com/JaidedAI/EasyOCR>.
9. *PyTorch* [online]. The Linux Foundation [visited on 2022-10-13]. Available from: <https://pytorch.org/>.
10. *Aura* [online]. FI MUNI [visited on 2022-12-02]. Available from: <https://www.fi.muni.cz/tech/unix/aura.html>.
11. DILMEGANI, Cem. *Best OCR: Benchmark on Text Extraction / Capture Accuracy* [online]. [visited on 2022-10-13]. Available from: <https://research.aimultiple.com/ocr-accuracy>.
12. NAKATANI, Shuyo. *Language Detection Library for Java*. 2010. Available also from: <https://github.com/shuyo/language-detection>.

BIBLIOGRAPHY

13. NAKATANI, Shuyo. *Language Detection Library* [online]. [visited on 2022-12-14]. Available from: <https://github.com/shuyo/language-detection>.
14. CVRČEK, Václav. *Korpus SYN2015* [online]. [visited on 2022-12-04]. Available from: <https://wiki.korpus.cz/doku.php/cnk:syn2015>.
15. TATMAN, Rachael. *English Word Frequency* [online]. [visited on 2022-12-04]. Available from: <https://www.kaggle.com/datasets/rtatman/english-word-frequency>.
16. *pandas* [online]. NumFOCUS, Inc. [visited on 2022-12-02]. Available from: <https://pandas.pydata.org/>.

A Running the final model

To run the final model, download the attached zip file and extract it. For just the model, we do not need the whole repository. The files in the *final-model* directory are sufficient. The instructions on how to run the model are also present in the *README.md* in the directory.

The whole repository is also available at our Faculty's GitLab¹.

A.1 Setup

Create a python virtual environment and install dependencies from *requirements.txt*. The implementation is tested on python version **3.9.15**. When running the model, the recommended python version is **3.9**.

```
$ virtualenv --python=<path/to/python3.9> venv
$ source venv/bin/activate
$ pip install -r requirements.txt
```

If the virtual environment is already created, activate the virtual environment.

```
$ source venv/bin/activate
```

A.2 Running the final model

To run the final model on a single image, run the following command from the command line.

```
$ python run_model.py <path/to/image>
```

A.3 Notes

Running the model via the command line is just for demonstration purposes. Running the model this way on more than one image would be unnecessarily slow. For those purposes, run the model programmatically, as shown in the *run_model.py* file.

1. <https://gitlab.fi.muni.cz/xvalcik1/image-flagger>

- Use import from *core_functions.py* import *
- Then call the function *pipeline_one_file(path, params...)* from code.

Running the model from the command line reinitializes the reader for every image, which adds a lot of processing time.

The function *pipeline_one_file(path, params...)* can be called with only one argument (as shown in the file *run_model.py*), using the default parameters described in the thesis. Still, the parameters can be changed by passing more arguments to the function.

A.4 GPU Optimizations

The model runs the best on GPUs. I ran all the performance-demanding tasks on FI's computational server, Aura, which has GPUs. The setup on a system with a dedicated GPU can be more complicated and specific to the host system. Most of the time, the setup described here should be enough.

B List of Electronic Attachments

- *image-flagger* - Repository with the source code of the implementation of the model.
- *final-model* - The final model extracted from the development repository.
- *um_results.csv* - The resulting CSV file for the Mathematics dataset.
- *ui_results.csv* - The resulting CSV file for the Informatics dataset.