Zurich University
of Applied Sciences

School of
Engineering

InIT Institute of Applied
Information Technology

Project work Computer Science

Document Digitization for Chess Scorecards

Author	Albin Abduli
Main supervisor	Prof. Dr. Mark Cieliebak
Date	18.12.2020

Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

DECLARATION OF ORIGINALITY

Project Work at the School of Engineering

By submitting this project work, the undersigned student confirms that this thesis is his/her own work and was written without the help of a third party. (Group works: the performance of the other group members are not considered as third party).

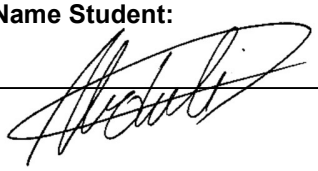
The student declares that all sources in the text (including Internet pages) and appendices have been correctly disclosed. This means that there has been no plagiarism, i.e. no sections of the Bachelor thesis have been partially or wholly taken from other texts and represented as the student's own work or included without being correctly referenced.

Any misconduct will be dealt with according to paragraphs 39 and 40 of the General Academic Regulations for Bachelor's and Master's Degree courses at the Zurich University of Applied Sciences (Rahmenprüfungsordnung ZHAW (RPO)) and subject to the provisions for disciplinary action stipulated in the University regulations.

City, Date:

Winterthur, 22.12.2020

Name Student:



Contents

1 Abstract

Chess games are noted by chess players at tournaments or in training on so-called "scorecards" in a standardized syntax. These scorecards vary in their structure for each occasion. To later analyse them with a computer program, the chess moves must be transferred manually to analysis software. Currently there exists an application in which an image of the scorecard is uploaded and the image is processed. The scorecard consist of a table and boxes in which a chess move is written. The application is able to detect the table, localise and extract the boxes where the move is written. The application uses the ABBYY Cloud OCR SDK API (ABBYY) for handwriting recognition, in which the image and the location of the individual boxes is sent. The result from the recognition then is post processed to validate the moves and the game. The application serves as a proof of concept and is developed by students Colin Dreher and Béla Horváth in their Bachelor Thesis [1]. The web application uses images of scorecards that are uploaded and a "Portable Game Notation" (PGN) file is retrieved. The application detects the chess moves on the scorecard and presents them to the user for control and correct a possible error and confirm the correct moves. In this thesis the web application was tested and analysed, there were detected possible areas that the application could be improved. The work was divided into small problems and tasks making it easier to compare the results. The main focus in the thesis is to refine the pre-processing algorithm where the location of the chess moves is detected and the box is extracted and to raise the quality of the image of the move sent to ABBYY for handwriting recognition. To raise the quality of the image, the issue of the characters that overlap in adjacent horizontal lines known as vertical overlapping is elaborated and a solution is implemented. The implemented parts of the algorithm were evaluated using a data set with different hand writings. The updated version of the application shows that an improvement of 24% is achieved. Overall, there is still a lot of possible areas where the application could be enhanced.

2 Preface

I am Albin Abduli, student in fifth semester bachelor in Computer Science at Zurich University of Applied Sciences. In this project work I had the opportunity to apply the knowledge gained during the studies and increase experience in software development, image processing, deep learning and artificial intelligence and improve personal skills.

I would like to thank Prof. Dr. Mark Cieliebak. Thanks to the good supervision and the great discussions, exciting topics and interesting suggestions and ideas discussed every week. I would like to thank Colin Dreher and Béla Horváth for the good documentation and suggestions in the bachelor thesis [1] of the project Very Chess of which this project work is based on.

3 Introduction

3.1 Current situation

Nowadays chess players write down the moves of a chess game in a Standard Algebraic Notation (SAN) on a scorecard while they are playing. Currently there is no digital representation of a scorecard. If the players want to analyse their game, they must replay all the moves by hand in a tool as for instance “Chess.com”. This takes a lot of time but can give the players valuable information about their behavior and tactics. In the thesis *Document Digitization for Chess Scorecards* [1] different software that exist on the market are discussed and compared and a web application "Very chess" is developed. In the corresponding application the following approach is defined:

1. Image alignment (align the uploaded image before continuing with the pre-processing)
2. Table extraction (filter noise in the scorecard and extract the table)
3. Chess move localisation (locate the relevant boxes that contain chess moves)
4. Recognize chess moves with Intelligent character recognition (ICR)
    a. Convolutional neural network (CNN) approach (pre-process and implement a custom CNN)
    b. ICR engine implementation (select and implement a fitting ICR engine/API)
5. Improvement of recognition (further improve the recognition of the implemented ICR engine)
6. Present PGN and download PGN (web interface to interact with the output and to download the output)

3.2 Application showcase

To show how the applications works, in this chapter a walk-through of the application on the users perspective will be presented. The workflow is based on the Bachelor thesis [1]. It starts with uploading an image to the web server and ends after the user decides to download the resulting PGN file as is shown in the Figure 1.

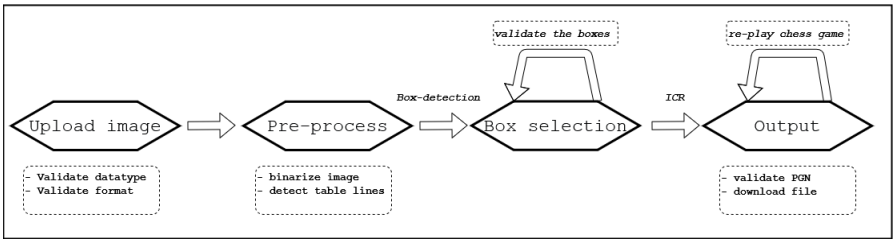


Figure 1: The workflow of the application. [1]

3.2.1 Upload image

First, with a press of the “Choose Scorecard” button an image of a scorecard must be chosen where all four corners of the sheet are visible. The image is uploaded with a press of “Digitalize Scorecard”. Only “.jpeg” and “.png” file formats can be uploaded. It is possible to directly take an image with a mobile phone. The resolution of the selected image must be above 2000x1500 pixel.

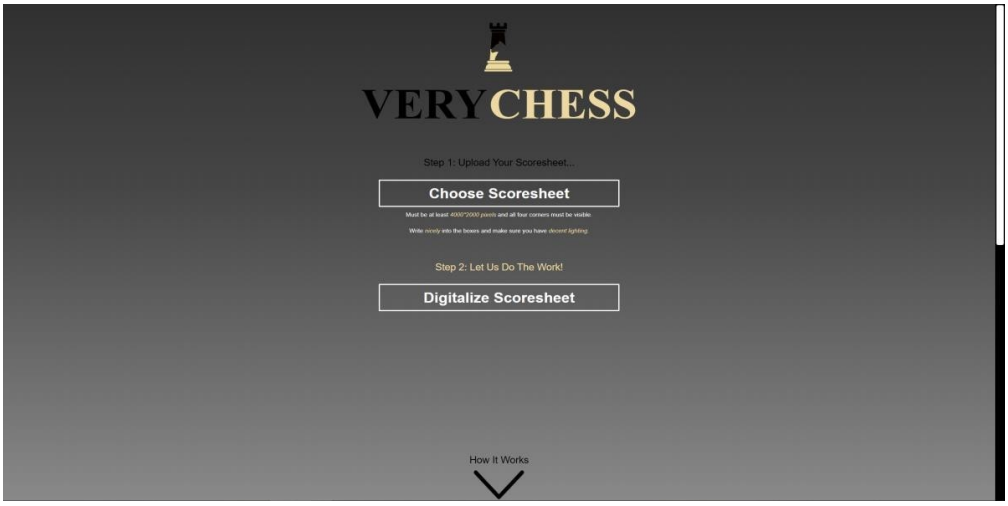


Figure 2: Landing page of the application where the user can upload an image and start the application. [1]

3.2.2 Box selection

In Figure 3, the user is asked to validate that the image alignment worked. This step can be continued when the “Confirm” button is clicked. If the image is aligned incorrectly, the user must take a new image with better quality.

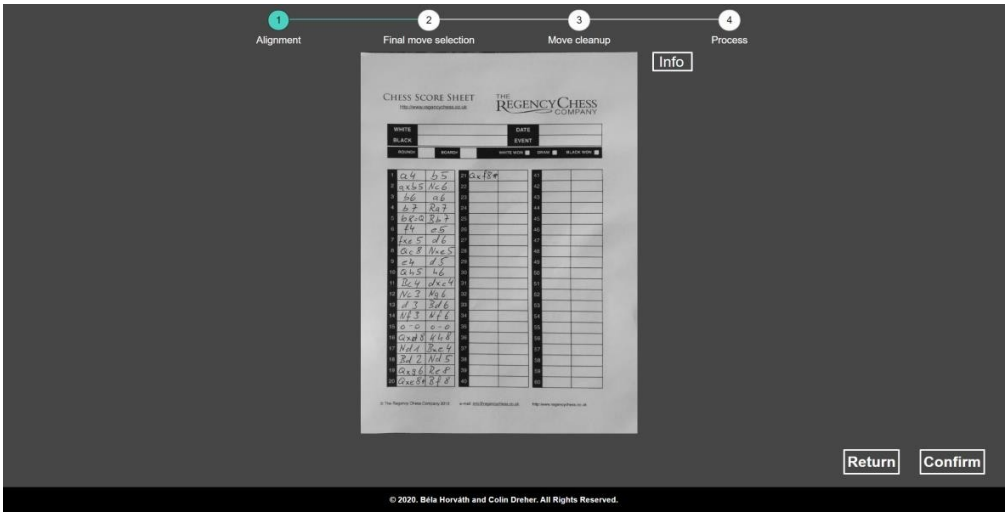


Figure 3: The aligned image is displayed in the page. The user can confirm the correctness of the alignment. [1]

After the first confirmation, the user is asked to select his “final move”. This corresponds to the last handwritten chess move on his scorecard. After the selection, the colours will indicate which parts are still active. With a click on “Confirm” the user is then asked to unselect any boxes that do not belong to his chess moves. This is only the necessary in certain scorecard layouts and thus not needed if everything looks correct as it is shown in Figure 4.

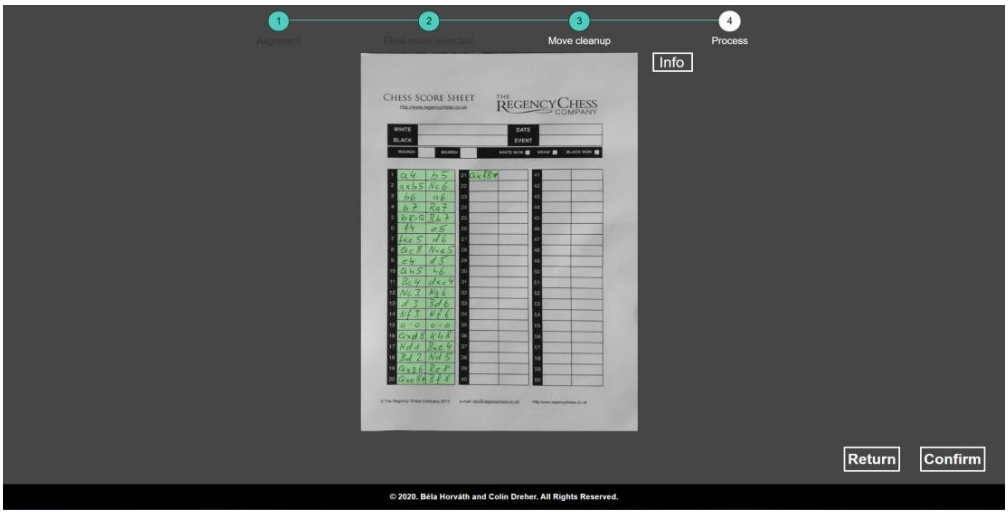


Figure 4: Unwanted boxes could be selected. This example does not contain any unwanted boxes. [1]

After validating that no wrongly highlighted boxes exist, the “Confirm” button can be pressed once more to send the selected boxes to the server. There they are recognized by ABBYY and based on the recognition a prediction is made for each move.

3.3 Output/replay

Once the moves are processed, the prediction is presented to the user in a table with the same layout as given on the scorecard.

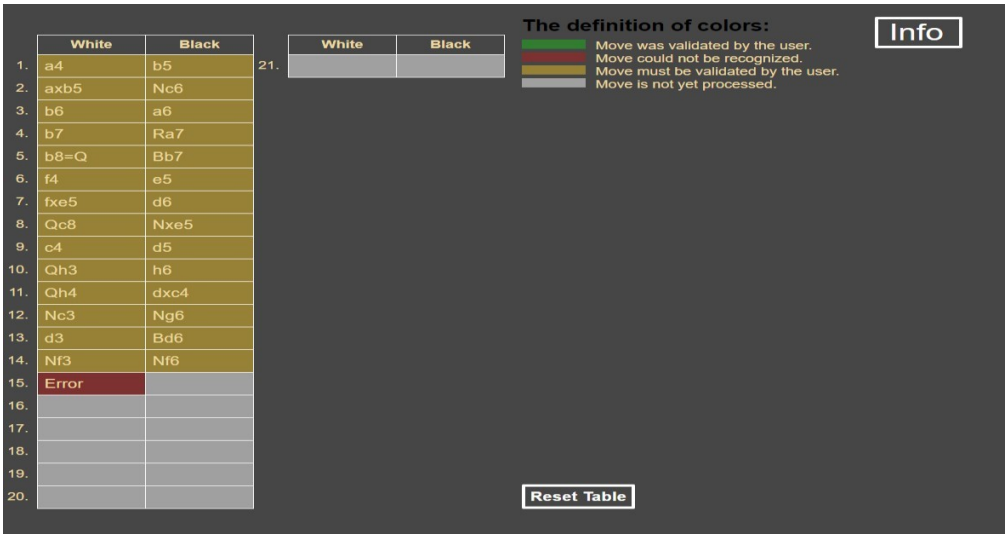


Figure 5: The predicted game that is presented to the user based on the recognition. [1]

The user has the possibility to control a prediction by clicking on one. A popup window displays an image of this box and the predicted move as shown in Figure 6. A every entry in the table has a colour and every colour represents a different state. For example, if the move is highlighted in green it corresponds to “Move was validated by the user” and it can be regarded as the true and thus correct move for this specific recognition (for this specific move). This state is reached by either changing the value or selecting one of the suggested candidates from the drop- down list. The correction can be applied with the “Apply” button.

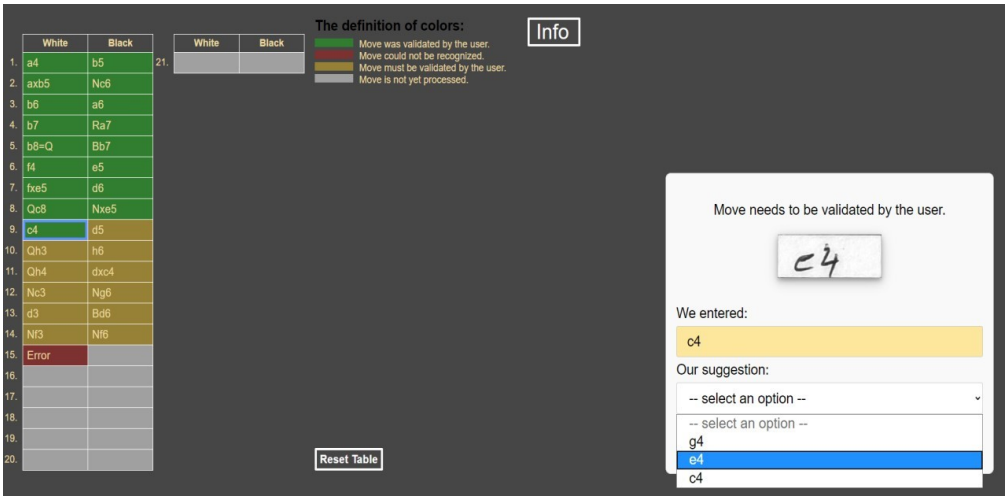


Figure 6: A popup window for the blue framed move “c4” in the table. The window shows the original image, what was predicted by the algorithm and the top suggestions. This move must be corrected to “e4”. [1]

If a correction is applied, the table data is sent to the server and processed with the corrections. Meanwhile the loading screen is shown. This process is repeated until all predictions are correct. If the user wants to reset everything to the original prediction, he can do so by pressing the "Reset Table" button. If everything is correct, the "Download" button appears as show in Figure 7.

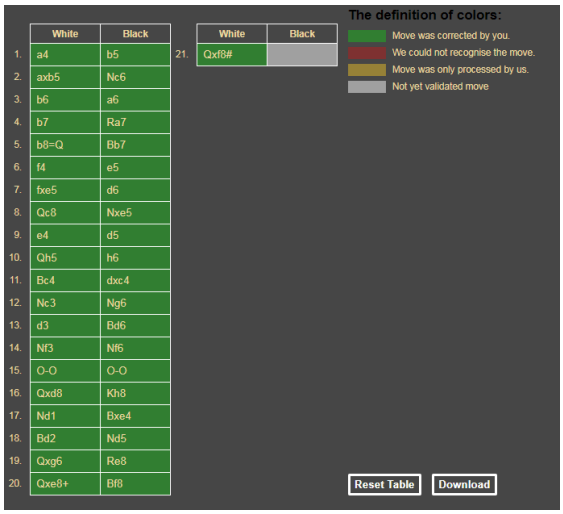


Figure 7: A correct game which is successfully predicted and validated by the user. [1]

By pressing the "Download" button, the user is presented with a form which he can optionally fill in (Figure 8). The form can be filled with metadata about the players, the tournament, and the winner to complete the PGN file. The user can confirm this form with the "Download" button to download a PGN file with the entered information and the processed game.

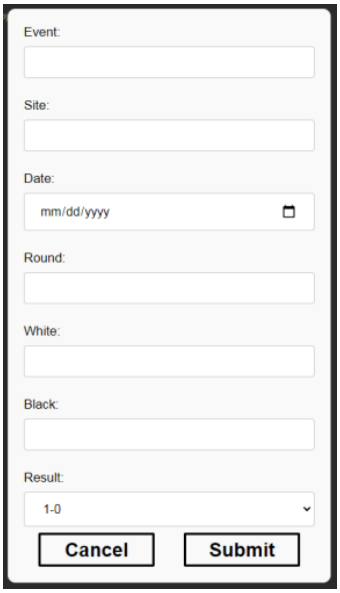


Figure 8: The final form to fill in the meta data for the PGN header. [1]

3.4 Possible improvements

After testing the first version of the application we have found different possible improvements in design and implementation that should be addressed and elaborated.

3.4.1 Overlapping characters in boxes.

The implementations of the image processing and text recognition algorithms as described on Application showcase are image alignment, table extraction, chess move localisation, ABBYY recognition and prediction of chess move candidates for improvement of recognition. During the process of chess move localisation and box extraction there are characters that overlap the box, in most cases bottom line as shown on the example on Figure 9.

Figure 9: Extracted box from the scorecard.

The image on the Figure 9 intends to extract the move "g3" which is written on the line number 2 on the Figure 10. Since the image is cut according the lines of the box, only the upper part of the character "g" is sent for recognition. This behaviour misleads the ABBYY OCR engine to the false result "a3". This issue occurs in the situation when a character most of the times "g" or "f" overlap with the the bottom line of the box.

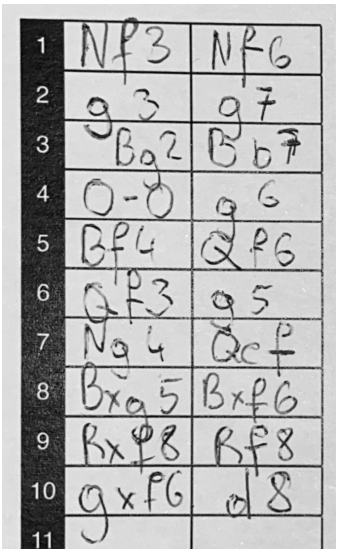


Figure 10: Scorecard with all the moves.

After image on the Figure 10 is sent to ABBYY for recognition, the move "g3" and "g2" on line number 2 are recognised as "a3" respectively "a2", the move "Bg2" on line 3 is recognised as "Ba2" and so on.

4 Goal and task

4.1 Main issue

In the past couple of months the application "Very Chess" has been analysed and improved. In this thesis, all the algorithm implemented to improve the application will be summarized and open points will be discussed. The implementation chapter will present the improvement that are made to the software. Finally, the achieved goals and open issues will be summarized again and an outlook on possible further developments will be given.

4.2 Addressing the issue

At the beginning, there were spotted possible further developments and improvement regarding the existing application. The main issue addressed in the thesis is Overlapping characters in boxes. Different approaches were considered during the development and the implementation was tested on the recognition data set containing 10 written scorecards from 6 different handwriting.

4.3 Motivation

In order to further develop the application the idea of improving the algorithms in the preprocessing section regarding overlapping characters in boxes was challenging and interesting. Overlapping of characters generally take place in two major instances. The first one is overlapping of characters placed adjacent to each other known as horizontal overlap see Figure 11 for example. The written move is "Ba2" the result of recognition is "a2".

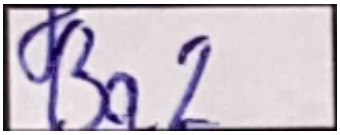


Figure 11: Horizontal overlap. The character "B" overlaps with "a".

and the second one is overlapping in adjacent lines or known as vertical overlap for example Figure 12. In this thesis the issue of the second instance will be elaborated.

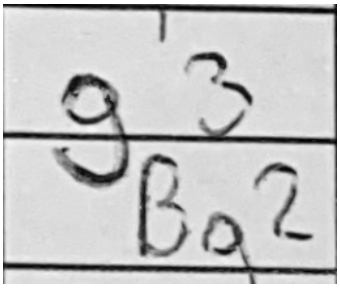


Figure 12: Vertical overlap. Character "g" overlaps the bottom line.

4.4 State of the Art

In order to solve the issue, we analysed which solutions already exist on the market and what approaches could be considered during the development. Most of the articles that were analysed addressed the problem of overlapping of characters placed adjacent to each other *A Survey on OCR for Over-Lapping and Broken Character's in Document Image* [2] and *Separation of touching and overlapping words in adjacent lines of handwritten text* [3]. However there are articles that address the problem of overlapping characters in adjacent horizontal lines *A Survey of Different Methods for Recognition of Overlapping Handwritten Text between Adjacent Lines of Text* [4]. According to the article [4] to extract the overlapping characters the following steps as shown in the block diagram on the Figure 13[4] and described below should take place.

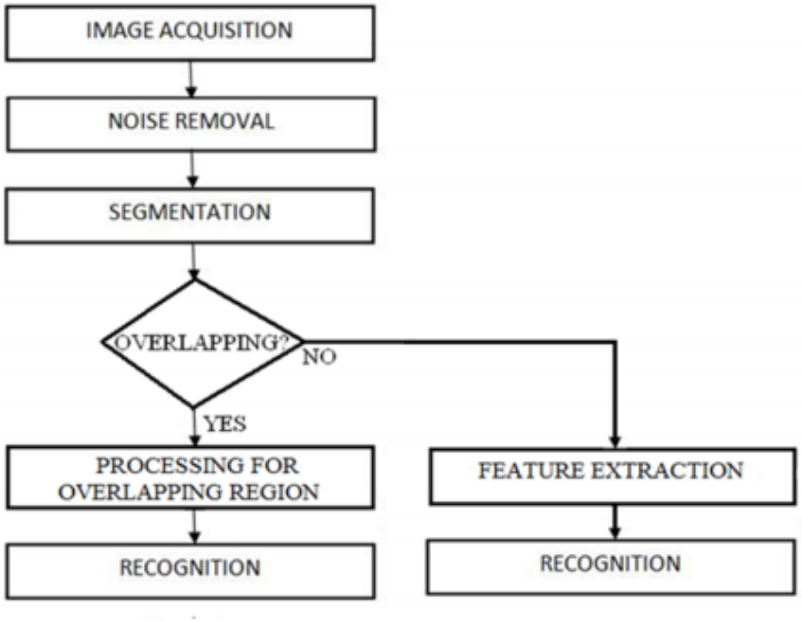


Figure 13: Block diagram of the system.[4]

- **Image Acquisition**

  To perform better in image processing, it is common practice to convert the multilevel image into a bilevel image of black and white. Often this process, known as thresholding, is performed on the scanner to save memory space and computational effort.

- **Noise removal**

  To eliminate noise from the input image, the connected pixels of very small area are deleted from the image by making use of morphological functions. A function called Morphological Dilation is used to fill the holes present in the image followed by Erosion to remove unwanted structures in the image.

- **Segmentation**

  Segmentation is one of the most important stages where the image is divided in three horizontal parts or segments and shown in the Figure 14[4].

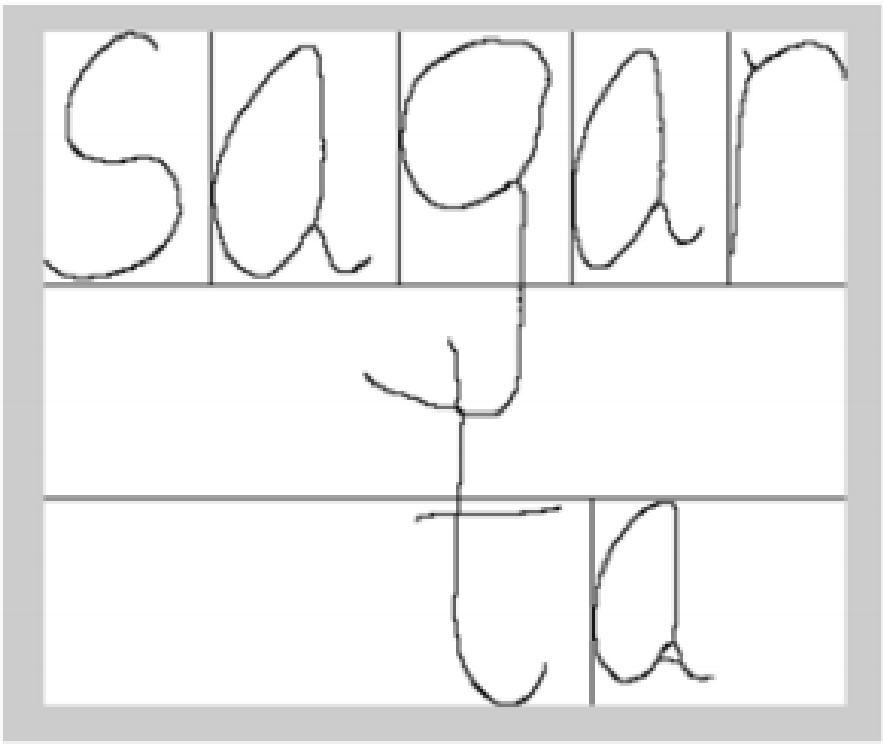


Figure 14: Horizontal and Vertical Segmentation to Individual Characters.[4]

- **Detection of Overlapping Region**

  A graphical representation of the tonal distribution in a digital image is plotted to show the number of pixels for each tonal value. If the histogram for the labeled overlapping component is non zero everywhere then overlapping or touching is present. In the other hand if the histogram for the labeled overlapping component is zero somewhere then the image is non-overlapping.

- **Recognition of Overlapped Characters**

  In this part of the article different methods are suggested for recognition of overlapped characters such as *Template Matching and Correlation Based Techniques* and *Feature Based Techniques*

- **Recognition of Non-Overlapped Characters**

  Non-overlapped characters are recognized and digitized by Decision Theoretic Methods. The principal approaches to decision-theoretic recognition are minimum distance classifiers, statistical classifiers and neural networks.

Another approach that was taken into account and integrated into the web application is *Remov-*

ing Horizontal Lines in image [5]. The tool is adopted and modified to raise the quality of the image during pre-processing phase. This approach consist of the following parts:

- Convert image to grayscale
- Otsu's threshold
- Create special horizontal kernel to detect horizontal lines
- Find contours on mask
- Repair image

The purpose of using the tool is to detect horizontal lines on the image, removing the line and repairing the broken characters that overlapped the horizontal line as is shown in the Figure 15. This approach will briefly explained in the Implementation section

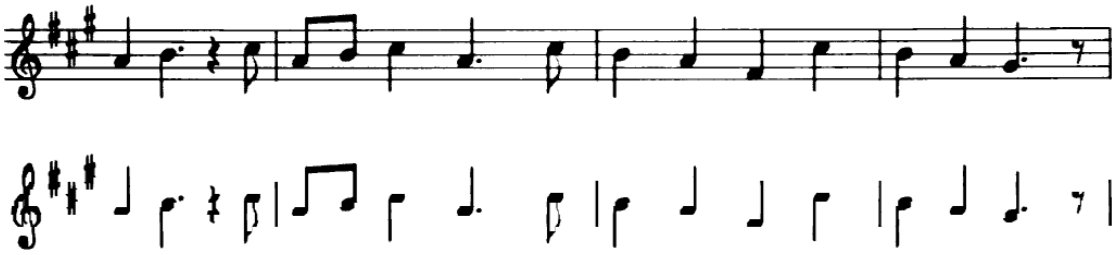


Figure 15: Music notes before and after line removal. [5]

5 Implementation

5.1 Initial Idea

Addressing the problem of vertical overlap of characters, the first step to get the complete character during the process of box extraction is to resize the height of the box by 50% of the next move in the vertical direction as shown in the Figure 16

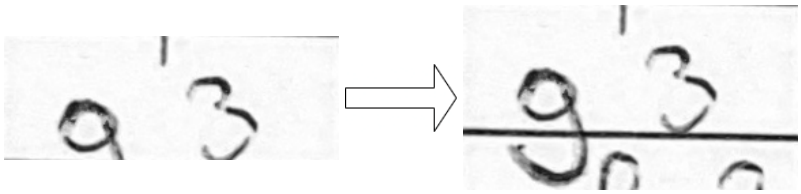


Figure 16: Resized box.

5.2 Removing the horizontal line

After resizing the image, to separate the characters from the horizontal line and create a cleaner image sent to ABBYY for better recognition, the algorithm for removing horizontal line is implemented and modified to achieve better results after recognition. The steps regarding this approach are defined in the following part:

A. Otsu's threshold

First step in the process of removing the horizontal line is Otsu's threshold. Otsu's threshold is used to perform automatic image thresholding. In the simplest form, the algorithm returns a single intensity threshold that separate pixels into two classes, foreground and background. The threshold is used to reduce the noise on the image and create a binary version of the image. The result after the threshold is shown on the Figure 17.

Figure 17: The image of the move after threshholding process.

B. Creating the base image

To obtain the base image of the move, which will be used later in the process, to remove the horizontal line, we repeat the Otsu's threshold and get the inverse image that is shown in the Figure 18.

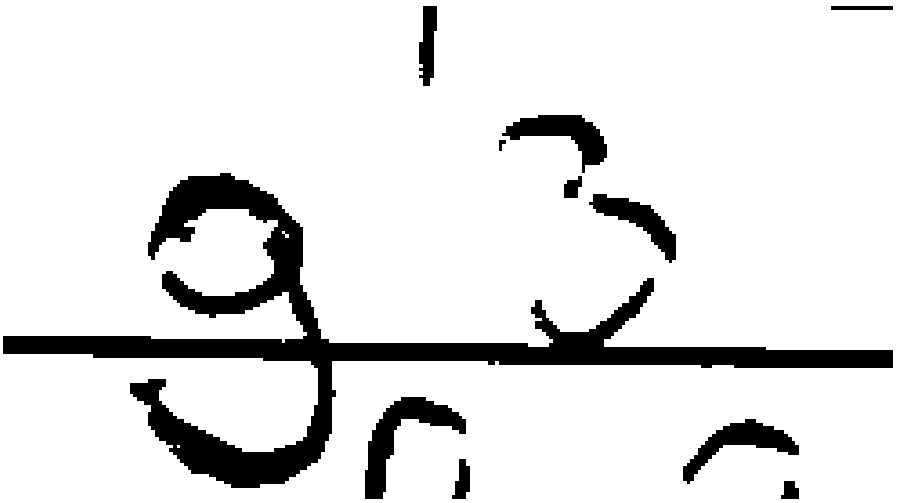


Figure 18: The base image of the move after inverting process.

C. Horizontal kernel to detect horizontal lines

To detect the horizontal line using image processing algorithms, a special rectangular kernel with predefined dimensions is used and morphological transformations are performed. The result is shown on the Figure 19, where the green line corresponds to the detected horizontal line.

Figure 19: Detected line in the move.

Disclaimer: *The image show on Figure 19 is just for understanding purposes, it does not take place in the process and its not part of the software.*

D. Find contours on mask

In order to distinguish the line from the whole image, the mask of the line is obtained from the thresh image Figure 17 after Otsu's threshold was applied and then the contours on the mask Figure 20 are located.

Figure 20: The image of the mask of the line.

E. Remove the horizontal line

After finding the contours on the mask, the next step is to remove the line on the base image shown in the Figure 18, and then the contours are redrawn and filled with white as is shown in the Figure 21.

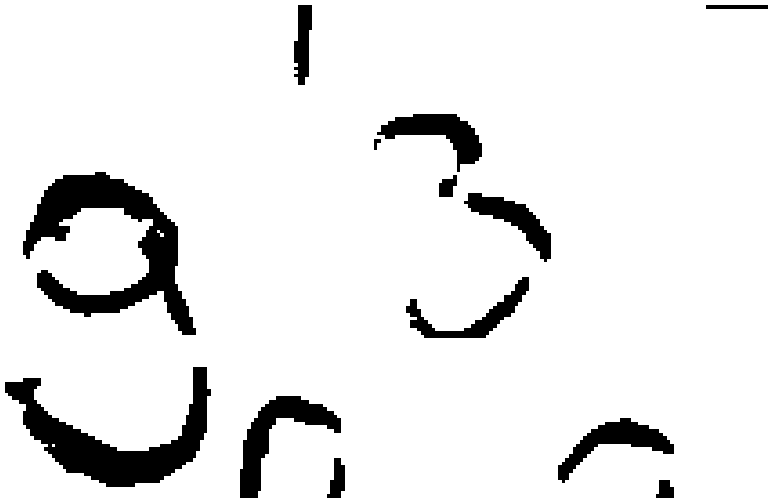


Figure 21: The base image without the line.

After implementing this algorithm, the problem of the broken character was introduced since ABBYY could not identify the characters yet because of the gap between two parts of the character separated by the line.

F. Repair the broken characters

To repair the broken characters and complete them in a single component two approaches were considered:

1. Morphological transformations to the whole image
   After applying morphological transformations to the whole image of the box using a vertical predefined kernel, the image was affected and the characters lost their form as is shown in the Figure 22.

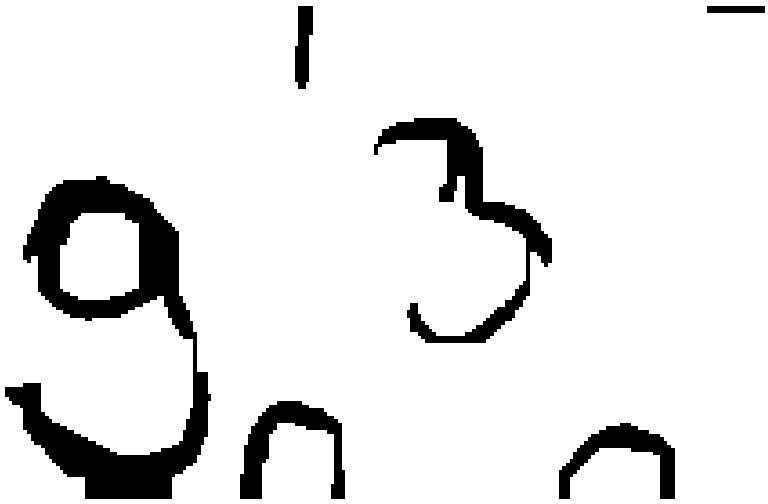


Figure 22: Image of the move after morphological transformations.

2. Morphological transformation to the removed line area
   The second approach to repair broken characters uses morphological transformations only to the area where the horizontal line is removed.

   a. Crop the horizontal area

   First the area is extracted from the image using the contours found on the mask Figure 23.

Figure 23: The image of the cropped part.

   b. Repair the character

   The morphological transformations using a vertical predefined kernel are performed Figure 24.

Figure 24: The image of the repaired image after crop.

c. Paste to the full image

The repaired part is then pasted to the base image to create a full character.

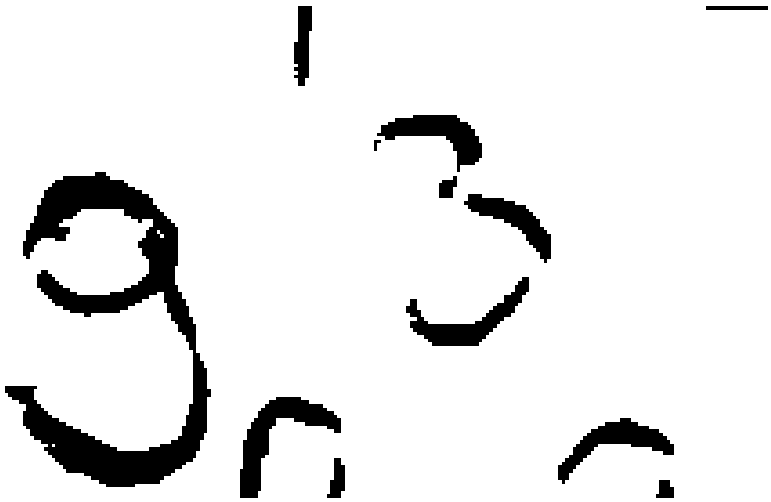


Figure 25: The resulting image after the repaired crop is pasted on the base image

G. Copy images of the processed moves to the scorecard image

After processing the individual images, they are copied and pasted in the scorecard image from which they are extracted from Figure 26. Then the aligned and processed image is sent to ABBYY for recognition.

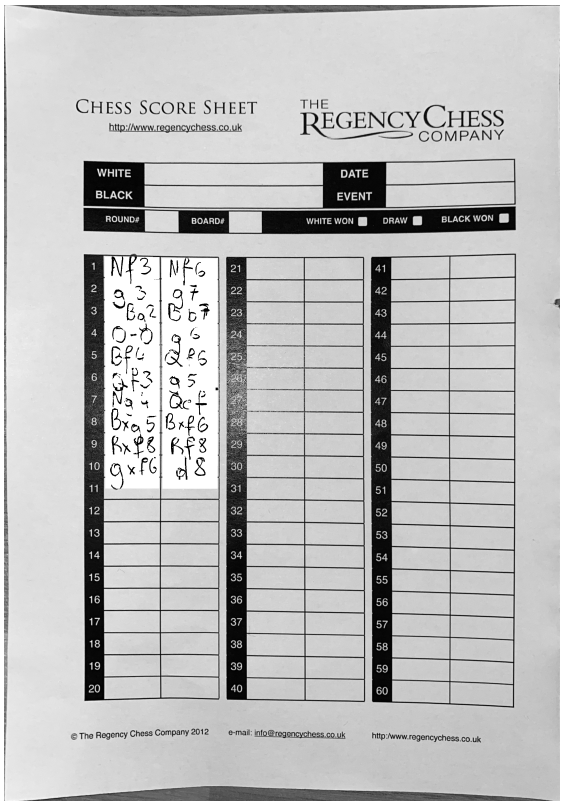


Figure 26: The resulting image after the repaired crop is pasted on the base image

Implemented architecture

The software architecture of the web application "Very Chess" is divided in two parts frontend and backend. The frontend depends on user input to either confirm or correct the presented recognition, computed in the backend. For the backend part the programing language Python is used, since it suitable for machine learning and image processing.

The implemented algorithm responsible for raising the image quality and improve the recognition are found on the following Python files on the PreProcessing section: preProcessImage.py, removeUnusedBoxes.py and on the frontendPipeline.py.

During the implementation of the algorithm different Python libraries are used such as OpenCV, PIL and NumPy. OpenCV is a free open source library used in image processing and NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. These libraries are used in thresholding process to converting the image in binary images and for detecting the drawing horizontal lines as segments using the functions findContours() and drawContours() from OpenCV.

Python Image Library (PIL) adds image processing capabilities. In the algorithm described in the thesis the module Image from PIL is used to crop parts of the image and paste them after processing.

5.3 Evaluations

The following evaluations are carried out with the aim of testing the individual steps and examining their improvement regarding the final prediction. For the following evaluations, a self-made data set and the recognition from the ABBYY API is used. In the test set 10 scorecards are used, examples are showwn in the Figure Figure 27. The moves on the scorecard do not present a valid game since they are intentionally chosen to be challenging for the recognition. The characters on the moves are purposely overlap the box border and test real world writings. The result of the testing corresponds only to the ABBYY recognition.

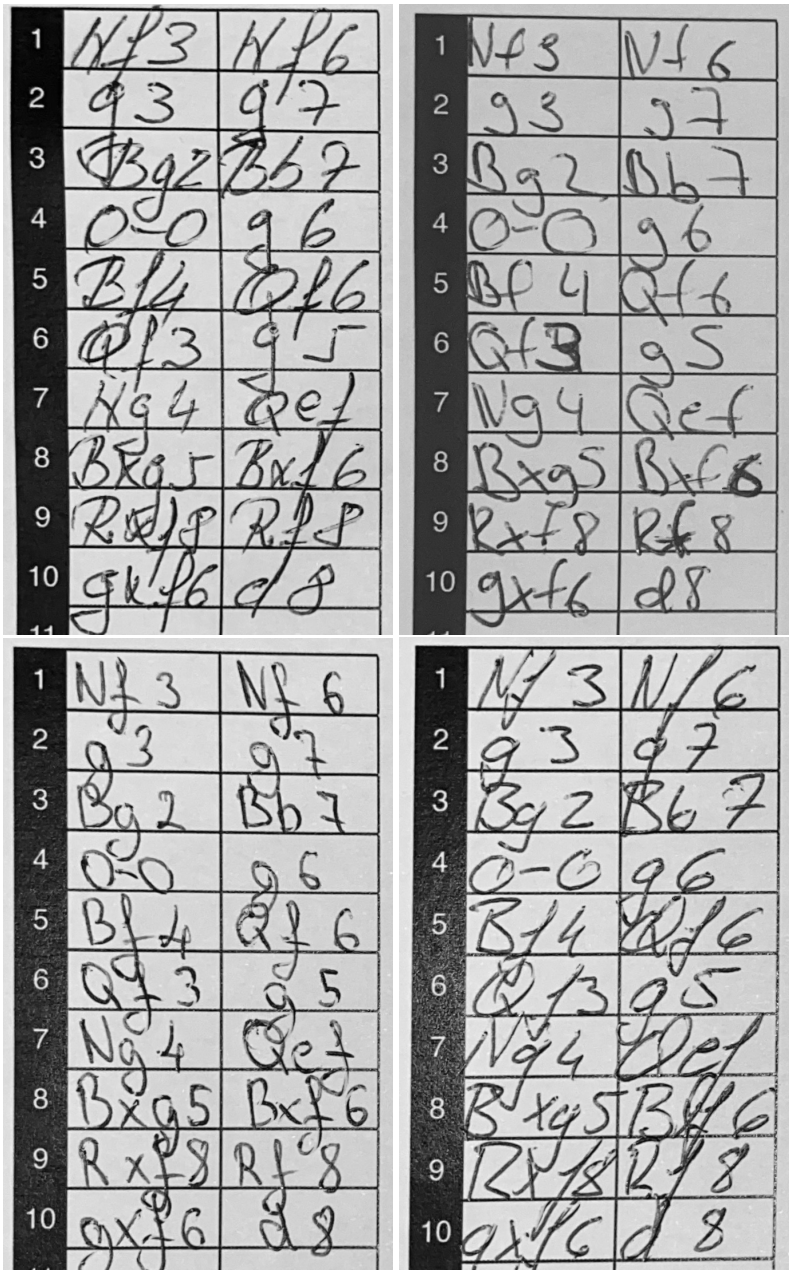


Figure 27: The test scorecard sheets. The images represent a focused part of the scorecards where the moves are written.

The moves used in scorecard for the test are presented on the Table 1. The move with number 14 does not correspond to a valid move, it is purposely written to test the regex sent to ABBYY for recognition.

Number	Move
1	Nf3
2	Nf6
3	g3
4	g7
5	Bg2
6	Bb7
7	0 - 0
8	g6
9	Bf4
10	Qf6
11	Qf3
12	g5
13	Ng4
14	Qef
15	Bxg5
16	Bxf6
17	Rxf8
18	Rf8
19	gxf6
20	d8

Table 1: Scorecard moves

The application is tested before and after the implementing the algorithm for improving the quality of the image. The evaluation of the test show 24% improvement from the first version of the application in which 136 errors were found and in the current situation only 104. Important is the fact that a regular and natural handwriting is used and no rules have been set in the process of filling the scorecards in order to have a realistic view of how the system works.

The ABBYY settings used to test and evaluate the recognition shown in the Table 2 are same for the initial version and for the improved version of the web application.

Parameter	**Description**
textType	Defined as “handprinted”, as the moves are handwritten.
writingStyle	Set to “german”, as the evaluated scorecards are written exclusively by German writers. This refers especially to the numbers 1 or 7 which have a different writing style in other languages
markingType	Set to "simpleText", after all the text is no longer inside of a box
letterSet	Defined to all occurring characters in the SAN moves: “RNBQKOabcdefghx12345678=+#-”
regExp	Set to a self-implemented regex, which only accepts syntactically correct SAN moves: “((((|([RNBQK](|[a-h])(|[1-8])(|x)|[a-h](|[1-8])x))[a-h]([1-8]|[18]=[RNBQ]))|O-O(|-O))(|[+#]|[+][+])”

Table 2: Defined settings for the recognition of handwritten moves.

In order to check the errors during the evaluations, a small tool programmed in Python is used. The tool is named ABBYYoutputEval.py and it is developed during the development of the first version of the web application by students Béla Horváth and Colin Dreher for evaluation purposes. The tool uses as input the correct moves which are written manually on the tool and the moves which are result from ABBYY recognition as an array for example [[[[’N’, 100]], [[’f’, 100]], [[’3’, 100]]], [[[’N’, 100]], [[’f’, 100]], [[’6’, 100]]], [[[’g’, 100]], [[’3’, 100]]], [[[’g’, 100], [’a’, 72]], [[’7’, 100]]], ...]. The brackets represent the hierarchy of the recognition where [’N’, 100] represents the recognised character "N" in a move and "100" represent the confidence of this character, the next order in the array hierarchy is [[[’N’, 100]], [[’f’, 100]], [[’3’, 100]]] which represents a move. The tool compares both version and calculates the number of wrong moves and creates a text file named eval_ABBYY.txt with the evaluation result as is shown in the example below Figure 28.

```
------------------------------------------------------------
Evaluation of file: sheet_2_improved_2
Amount_errors: 6
Bg2 - Kba2
Qf3 - Nf3
Ng4 - a4
Qef - Qc1
Rxf8 - Kxf8
gxf6 - axf6
------------------------------------------------------------
```

Figure 28: Evaluation result of ABBYY recognition.

6 Conclusion

The goal of this thesis was to analyse the web application "Very Chess" and algorithms in the image processing phase which are responsible for possible errors in the detected moves, and optimize the corresponding algorithms. Different ideas to refine the image of the extracted boxes are discussed. In the thesis the problem of overlapping characters with the horizontal line in the bottom is elaborated.

The algorithm responsible for the pre-processing phase is optimized to improve the quality of the individual images of the moves, thus the number of correct moves in the recognition process is increased. The algorithm makes a cleaner version of the image of individual boxes. The idea is to detect the the characters overlap with the horizontal line in the bottom of the box, remove the horizontal line and connecting the broken characters. To develop the application, different image processing and deep learning algorithms are implemented. To test the application a test set with 10 written scorecards from 6 different hand writings are used and an improvement 24% is achieved. The algorithm could also be used if in later development another recognition tool or custom CNN is implemented.

7 Further development possibilities

To further improve the application, the custom recognition tool could be investigated and implemented. A custom recognition tool might reach a higher confidence score than ABBYY. The underlying algorithms may be improved to raise the quality of the image using segmentation to detect different objects in the image and remove the not important one. The goal would be to have a clear image as shown. in the Figure 29

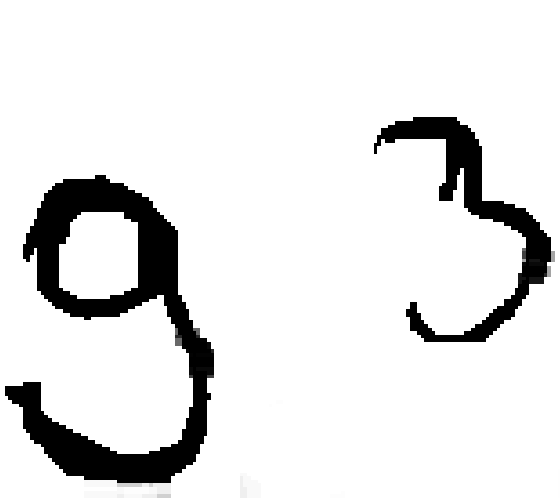


Figure 29: Ideal image of the move.

Since each step in the implementation offers potential for optimization, a documented test run of the application should be carried out to identify where the most errors occur. Once these points of failure are identified, they can be fixed successively.

In order to get a better recognition result, it is considered to create an image where different versions of the individual boxes are saved and sent to ABBY for recognition. The recognition then needs to be post-processed and the result with the biggest confidence could be selected, the layout of this idea is shown in the Figure 30.

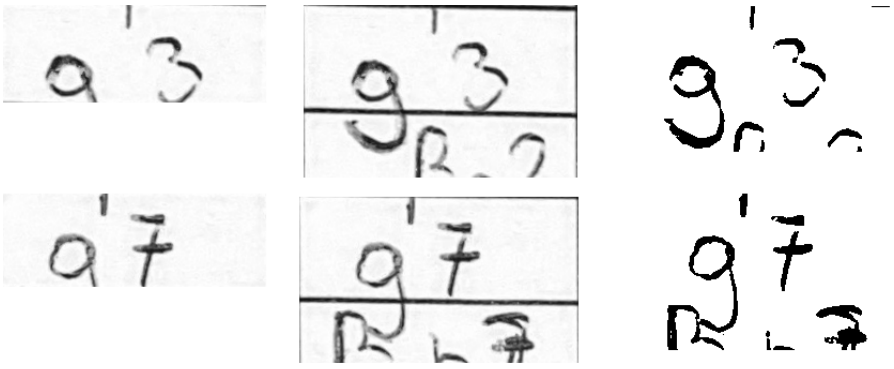


Figure 30: Image of combined image versions of a move.

Based on the Bachelor Thesis [1] to develop a commercially viable product, it requires a solid foundation and additional features:

- The frontend application should be ported to a current web framework.
- The architecture of the application should be adapted to function like an API to make it more extensible.
- The application in its current state only allows single user access and no user management system is in place. This extension would have to be installed to handle user accounts, user specific data and confidential information.
- Since the ABBYY OCR engine is not free of charge, an alternative and free approach would be beneficial to lower the cost of the application and allow for further specialization of the recognition.
- Further layouts could be supported within the application through focusing on the next most common layout apart from solid tables.
- Include chess analytics into the application and create an all-in-one solution to digitize and analyse chess games.

8 Technical documentation

This part describes the full installation process to locally develop or to deploy the app on a server. Since the technology is not changed the documentation is based on the thesis [1].

8.1 Local installation

Repository data:

Fork the corresponding repository *The repository will be given by the supervisor*

Clone the repository locally or copy the files into a folder and create a new repository.

ABBYY Setup:

Create a new account: https://www.ocrsdk.com/

Check your E-mail for the Application Password. Copy the password.

from: https://cloud.ocrsdk.com/Account/Welcome copy your Application ID in a similar format "551ff3ed-40d8-4f0f-8d2b-7150d25861de"

Open AbbyyOnlineSdk.py file located in "./Algorithm/ABBYY_OCR/AbbyyOnlineSdk.py" and change the ApplicationId and Password accordingly inside AbyyOnlineSdk class.

This allows you to use the free 500 pages. If a licence is in place you must use the account details of the licenced account.

Backend Setup:

1. Install Python version 3 (sudo apt-get install python3.X)
   During the development of the web application the Python version 3.8.0 is used
2. Install pip3 python3-pip (sudo apt-get install pip3)
3. Install python virtualenv: pip install -upgrade virtualenv
4. Create and activate the venv in the app folder

   For Windows (Python v. 3.8):

   `cd ‘‘appname’’`

   `virtualenv -python ‘‘c:\python38\python.exe’’ env .\env\Scripts\activate`

   PyCharm:

   Create virtualenv in interpreter settings. Activate venv for the project

   ---

5. Install requirements:

   pip3 install -r requirements.txt

8.2 Deployment

Follow:

- https://apu.cloudlab.zhaw.ch
- https://www.youtube.com/watch?v=YFBRVJPhDGY
- https://stackoverflow.com/questions/39418012/my-apache-wsgi-flask-web-app-cannot- import-its-internal-python-module

to have as a backup when configuring the mod_wsgi. You can also choose another web server and gateway of choice like Nginx or similar.

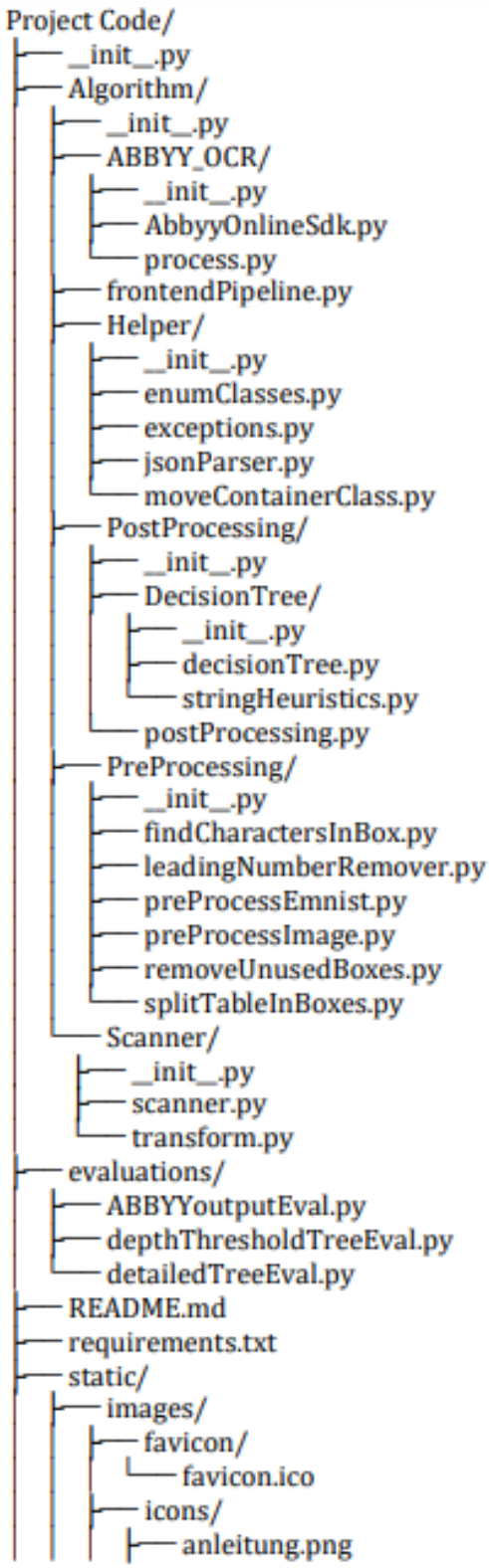
Installation

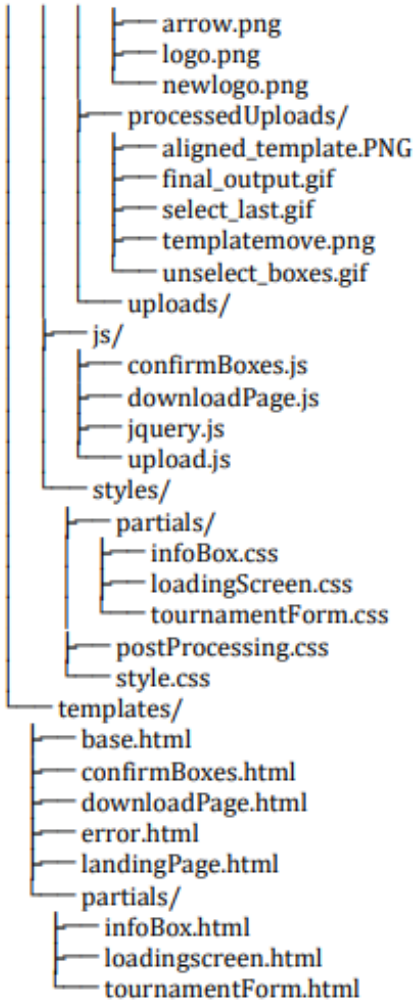
1. Set up server like documented on cloudlab.
    - Forward port 80 (or 443 if needed) to the server.
2. Install Python and Pip (sudo apt-get install python)
    - Forward port 80 (or 443 if needed) to the server.
    - sudo apt update
    - sudo apt install software-properties-common
    - sudo add-apt-repository ppa:deadsnakes/ppa
    - sudo apt install pythonX.Y (<– enter version)
    - sudo apt install python3-pip
3. run: sudo pip3 install -r requirements.txt (located in base of repository)
   UBUNTU ONLY
    - sudo apt install tesseract-ocr (use this if it cannot run because of tesseract ocr error)
4. Follow: https://www.youtube.com/watch?v=YFBRVJPhDGY tutorial on how to deploy with apache wsgi.
    - https://stackoverflow.com/questions/39418012/my-apache-wsgi-flask-web- app-cannot-import-its-internal-python-module answer to configure the YOURAPPNAME.wsgi file, otherwise it will not work correctly.

   NOTE:
5. Connect to the server via the public IP-Address and use your page!

8.3 Overview of the files

The following tree structure shows the whole application:




arrow.png
logo.png
newlogo.png
processedUploads/
aligned_template.PNG
final_output.gif
select_last.gif
templatemove.png
unselect_boxes.gif
uploads/
js/
confirmBoxes.js
downloadPage.js
jquery.js
upload.js
styles/
partials/
infoBox.css
loadingScreen.css
tournamentForm.css
postProcessing.css
style.css
templates/
base.html
confirmBoxes.html
downloadPage.html
error.html
landingPage.html
partials/
infoBox.html
loadingscreen.html
tournamentForm.html

Glossary

Box:	A cell in the table layout of the scorecard which represent a chess move.
ICR:	Intelligent character recognition is an advanced optical character recognition or rather more specific handwriting recognition system that allows fonts and different styles of handwriting to be learned by a computer during processing to improve accuracy and recognition levels.
Move:	Chess move, the act of moving a chess piece or a player's turn to take some action permitted by the rules of the game
OCR:	Optical character reader (OCR) is the electronic or mechanical conversion of images of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo (for example the text on signs and billboards in a landscape photo) or from subtitle text superimposed on an image.
PGN file:	Portable Game Notation is a standard plain text format for recording chess games, which can be read by humans and is also supported by most chess software.
PNG:	Portable Network Graphics is a raster-graphics file format that supports lossless data compression.
SAN:	Standard algebraic notation (SAN) is a system for recording chess moves. Moves are represented by the name of the piece and the square to which it is being moved.
Scorecard:	A score sheet is a tool used to record the moves played by both players during an over-the-board (OTB) chess game. It is usually a sheet of paper that contains multiple fields for a player to add relevant information about the game being played.
Tool:	A programming tool or software development tool is a computer program that software developers use to create, debug, maintain, or otherwise support other programs and applications.

Bibliography

[1] Béla Horváth, Colin Dreher: *Document Digitization for ChessScorecards* Bachelor's Thesis [Date: June 19, 2020]

[2] A. K. Gaur, D. S. Bharangar and M. C. Trivedi: *A Survey on OCR for Overlapping and Broken Characters in Document Image: Problem with Overlapping and Broken Characters in Document Image* [Online]
Available at: `https://ieeexplore.ieee.org/document/7065461` Published at: 2014 International Conference on Computational Intelligence and Communication Networks [Published: November 14-16, 2014]

[3] Kalyan Takru, G. Leedham: *Separation of Toching and Overlapping Words in Adjacent Lines of Handwritten Text* [Online].
Available at: `https://www.researchgate.net/publication/262332575_Separation_of_Toching_and_Overlapping_Words_in_Adjacent_Lines_of_Handwritten_Text` [Published: August 2002].

[4] LRaj Sheth, Ketan Patil, Niharika Thakur, Prof. K.T.Talele. *A Survey of Different Methods for Recognition of Overlapping Handwritten Text between Adjacent Lines of Text* Sardar Patel Institute of Technology [Online].
Available at: `https://www.spit.ac.in/wp-content/uploads/profkttalele/project/2011-12/Overlapping%20Character/Survey%20Paper.pdf` [Accessed: December 14, 2020].

[5] *Removing Horizontal Lines in image (OpenCV, Python, Matplotlib)* [Online].
Available at: `https://stackoverflow.com/questions/46274961/removing-horizontal-lines-in-image-opencv-python-matplotlib` [Accessed: December 14, 2020].

List of Tables

List of Figures