Vrije Universiteit Amsterdam

Bachelor Thesis

Verelect: Make Elections Transparent Again

Author: Tadas Majorovas (tms226)

1st supervisor:	Prof. dr. ir. H.J. (Herbert) Bos
daily supervisor:	Drs. Johannes Blaser
2nd reader:	Asst. prof. dr. E. (Erik) van der Kouwe

A thesis submitted in fulfillment of the requirements for the VU Bachelor of Science degree in Computer Science

July 10, 2024

"The computing scientist's main challenge is not to get confused by the complexities of his own making."

- Edsger W. Dijkstra

Abstract

Elections need to be transparent to ensure public trust in the democratic process and prevent fraud. The current electoral system in the Netherlands fails to be transparent. Every election generates a large amount of data, making it very difficult to manually sift through it without automated tools. This thesis aims to create a system designed to automatically parse scans of documents, which indicate vote total per polling station, to allow the average person to verify election results faster and easier. To do this, deep-learning models and OCR engines are used to identify and extract vote totals for each candidate. This information then gets compared with the official outcome. Various systematic policy changes are also proposed. They are designed to ensure that data produced by the election is more widely available in good quality.

Contents

CONTENTS

1

Introduction

In a true and open democracy, elections are a crucial staple. They allow citizens with voting power to choose their representation or to show which policies they support. The essential facet that legitimizes this process is the voter's trust in the validity and fairness of the results. A lack of trust can call into question the legitimacy of the results and, thus, the government itself. For this reason, every true democracy must host the election in a way that bolsters this trust. One way to achieve this is by designing the election process to be transparent and verifiable by anyone.

This thesis focuses, for the most part, on Dutch elections. However, the methods and systems described could apply to similar electoral systems. Under the current system, the Dutch election is mainly paper-based. The voting ballots, calculated totals, process documentation, and correction forms are all on paper. A volunteer manually enters the outcomes into a computer after these paper documents are filled out and submitted. Later, these forms are scanned and publicly made available online. At first glance, due to the extensive paper trail this process leaves behind, it might look like the system offers strong transparency. A big caveat is that a single election produces hundreds of thousands of pages worth of documents. Because of the tremendous volume of information for the average voter, this system yields little to no impact on overall transparency. The average person cannot manually sort through all of this data to double-check the accuracy of the results. Because of this, the Dutch electoral system could risk losing the favor and trust of their voters.

This research aims to contribute to the general validity and trustworthiness of the election by creating a system that allows a user to quickly verify the results of the Dutch election without the need to comb through many documents. To deal with this problem, this thesis introduces Verelect [1]. Verelect uses deep learning methods to process various scans from

artifacts generated by the election so that they can be automatically cross-checked against the official results. The main goal of this system is to reduce the amount of forms a person has to check by handling all of the easy cases automatically. This allows auditors to focus on forms that are more likely to include a mistake.

The overall goal for Verelect is to allow people to find mistakes in the vote counting of the election. In an ideal scenario, Verelect could acquire election documents from online sources and identify each candidate's accumulated votes with minimal human interaction. The system could then use this information to determine whether there were any mistakes in the officially reported numbers. Once discrepancies are found, a human could manually investigate the potential error and determine the cause of the situation. Documents verified to match the official outcome could safely be skipped.

This thesis also proposes concrete improvements to the Dutch election process and the documents used based on the challenges encountered while designing Verelect. These proposals aim to, by design, make the system more transparent and benefit tools like Verelect by making election documents more straightforward to work with overall.

This thesis contributes the following:

1. A labeled dataset of pages containing vote totals with classes for tables, party names, subtotals, and totals.
2. A labeled dataset of rows with data in vote total tables.
3. The Verelect system.
4. A proposal to make election data more consistent.
5. A proposal for centralized hosting of election data.
6. A proposal and design for a new standard of form to optimize the performance of OCR and deep-learning models.

2

Background

The Dutch election has several different formats depending on what is being voted on. This thesis only focuses on the House of Representatives election of 2023. Even though newer elections have taken place, this was the most recent at the time of the design and implementation of Verelect. Despite this, the methods described in this thesis should be applicable universally with minimal to no adjustments needed.

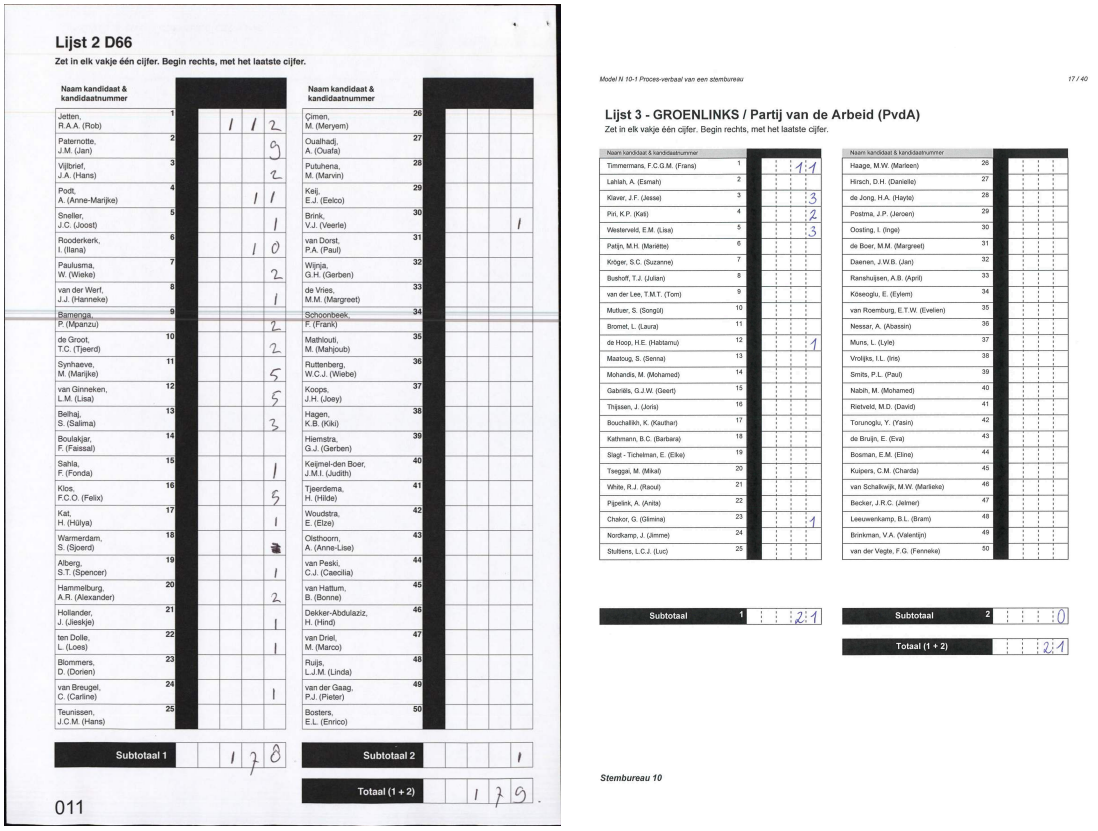
2.1 Types of forms

There are two types of vote counting in the Dutch election: Decentralized voting and centralized voting. For the purposes of Verelect and this thesis, the only difference that matters is that these systems use a different type of form to count the vote totals. Decentralized stations use Model N10-1 forms, while centralized ones use Model Na 31-2[2]. Both of these forms contain pages with tables containing vote totals. Figure 2.1 shows two of these pages. Each page includes information on which party they relate to and which party member got how many votes. Other differences exist but are irrelevant and will be excluded for ease of explanation.

2.2 Publishing

After the vote counting has concluded, these documents get scanned and uploaded to the municipality website for public viewing. Unfortunately, while these documents are public, they are not easily found. No user-friendly interface or API is provided to access these documents. Due to the papers being scanned and having no metadata outside of filenames, these documents are barely available on various search engines. To make matters worse,

2. BACKGROUND



(a) (b)

Figure 2.1: Example of filled-in forms released by (a) Amsterdam polling station 11 and (b) Lopik polling station 10, respectively

instead of hosting the files, some municipalities opt to host them on 3rd party platforms. This makes the documents significantly more challenging to find and recognize since 3rd party platforms have randomized URLs and sometimes even file names. Hence, tracking which polling station issued which forms is often problematic.

If a mistake is made in the counting process and not caught early enough, a "Corrigendum" form is issued. These forms are similar to the ones shown in figure 2.1, but show only the corrected vote totals. In a corrigendum table, rows with errors are rewritten, and others are omitted. Because of this, both forms are needed to understand the results thoroughly. These forms are published alongside their original counterparts.

The official results are published in an EML format [3] on the government's website. The EML format is not widely supported by various tools. To help with that, an open-source tool called eml2sql published by the Kiesraad converts all election data into a single easy-to-use SQL file [4], most likely for legacy reasons.

The EML format consolidates data from multiple sources into a single file, aggregating results from individual polling stations, municipalities, and districts. This creates a system where results can be verified at each level. In theory, if you sum up all of the polling stations in a municipality, the municipality's results should equal the ones of the summed-up polling stations. In practice, this is not always the case since clerical errors where some documents are not sent in on time will result in various discrepancies. For instance, in the 2023 House of Representatives, a number of municipalities did not send in their N14-1 forms in time, causing outdated polling station totals to be published.

2.3 Deep-learning

Software must first understand the layout of the scanned documents to understand what is written on them. Alongside the vote totals, essential elements on the page, like party names, tables, candidate numbers, totals, and subtotals, need to be inferred. Without these elements, vote totals provide no helpful information. Additionally, vote totals on these forms are written by hand, which poses a challenge since human error from many different writers introduces a lot of variance. It is commonplace to employ various deep-learning concepts to detect and extract from scans of documents [5]. It is highly accurate and efficient with various unstructured data types like images. One downside of this approach is that it requires a lot of manually labeled data to train, and, unfortunately, a dataset containing this data is yet to be created.

In the context of machine learning, generational models are a type of algorithm that improves solutions over several rounds or generations. Each generation chooses and combines the best solutions to create new, potentially better solutions. An epoch is one complete pass through all the training data. During each pass, the model learns and adjusts to improve its accuracy. Multiple epochs are usually needed to effectively train a model, with each one helping to enhance the model's performance. The epoch with the best performance is picked at the end of the training.

3

Overview

To achieve the goal of semi-automating election verification, Verelect needs to be able to recognize form layout. Just being able to read pure vote totals is not enough since that does not provide any information on where these votes came from and to which candidate they belong. Verelect needs to acquire election data, recognize which page is relevant, and identify and extract information on which party the page belongs to and how many votes each candidate got. These tasks are sequential to one another. That is why Verelect is made of several modules, each responsible for its own task, which later provides input for their successive module. Figure 3.1 shows a chart showing these modules. The modules in Verelect are Form Downloader, Form Element Model, Data Row Model, OCR, EML Parser, and Results.

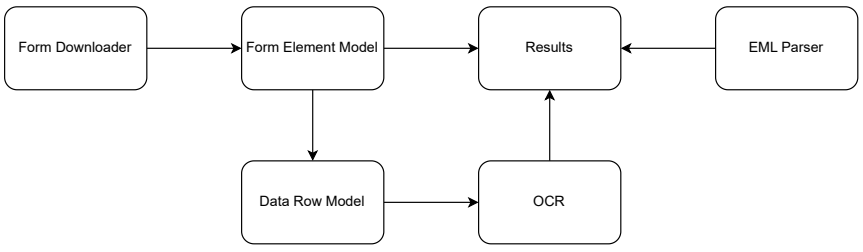


Figure 3.1: Diagram of the Verelect system

As the Dutch Elections generate a significant amount of data, minimizing the number of pages the system needs to parse is essential to optimize compute times. That is why each module is designed to discard data that is either unneeded or malformed enough not

to provide any valuable data. Discarded data is marked as not parsed to be prioritized for human review.

The first step Verelect takes is to download all relevant data and sorts it based on which polling station the documents were issued by. Once all of the data is gathered, Verelect uses the Form Element Model to parse relevant pages of documents and get their elements, namely party names, tables, subtotals, and totals. The identified tables undergo a separate processing step to optimize OCR result reliability, where each row with data is extracted with its index. Once all the relevant data is detected, elements get passed through an OCR engine to extract the printed and handwritten text. Once the text is analyzed, the results are compared to those published in the EML files. The software outputs a CSV file containing the results. An example excerpt of one of the outputs can be found in figure 3.2. The results are highly granular, capturing which municipality, polling stations, party affiliation, and which candidates these votes belong to. The output includes a status column showing if the data was validated successfully or why the file was not validated.

```
Leiden,Leiden,47,3,1,119,119,Matches
Leiden,Leiden,47,3,2,47,47,Matches
Leiden,Leiden,47,3,3,32,32,Matches
Leiden,Leiden,47,3,4,3,3,Matches
Leiden,Leiden,47,3,5,18,18,Matches
Leiden,Leiden,47,3,6,0,0,Matches
Leiden,Leiden,47,3,7,1,1,Matches
Leiden,Leiden,47,3,8,2,2,Matches
Leiden,Leiden,47,3,9,0,0,Matches
Leiden,Leiden,47,3,10,0,0,Matches
Leiden,Leiden,47,3,11,1,1,Matches
Leiden,Leiden,47,3,12,1,1,Matches
Leiden,Leiden,47,3,13,39,39,Matches
Leiden,Leiden,47,3,14,0,0,Matches
Leiden,Leiden,47,3,15,0,0,Matches
Leiden,Leiden,47,3,16,1,1,Matches
Leiden,Leiden,47,3,17,3,3,Matches
Leiden,Leiden,47,3,18,4,4,Matches
Leiden,Leiden,47,3,19,2,2,Matches
Leiden,Leiden,47,3,20,2,2,Matches
Leiden,Leiden,47,3,21,1,1,Matches
Leiden,Leiden,47,3,22,2,2,Matches
Leiden,Leiden,47,3,23,2,2,Matches
Leiden,Leiden,47,3,24,0,0,Matches
Leiden,Leiden,47,3,25,1,1,Matches
Leiden,Leiden,47,3,26,3,3,Matches
Leiden,Leiden,47,3,27,7,7,Matches
Leiden,Leiden,47,3,28,0,0,Matches
Leiden,Leiden,47,3,29,0,0,Matches
Leiden,Leiden,47,3,30,3,3,Matches
Leiden,Leiden,47,3,31,1,1,Matches
Leiden,Leiden,47,3,32,0,0,Matches
Leiden,Leiden,47,3,33,2,2,Matches
Leiden,Leiden,47,3,34,3,3,Matches
```

Figure 3.2: Example excerpt from the outputs of Verelect for Leidens polling station 47. Data is listed in format district, municipality, polling station, party, candidate, expected vote total, detected vote total, status

Use case For the user, Verelect serves as a first-pass filter. It filters out pages and rows in the documents that are highly likely to be correct. This allows the user to focus their efforts on cross-referencing other data with a greater chance of containing a mistake. Verelect can reliably detect, parse, and check cleanly written pages with common styling. It is also proficient at checking and filtering empty pages where all candidates received no votes. Verelect still struggles and requires human intervention on pages with messy handwriting or nonstandard styling.

Roadmap In chapter 4 the design of the Verelect system is discussed and analyzed. In chapter 5, implementation details are provided on Verelect. In chapter 6, the system performance is tested and evaluated. In chapter 7 Verelects results are discussed and analyzed. In chapter 8 problems that hamper Verelects performance are discussed. In chapter 9, a proposal was discussed to make significant changes to the election process that would give tools like Verelect more reliable performance.

4

Design

4.1 EML

EML files can be freely downloaded from the Dutch government's website. While Verelect loads the EML data, a check counts the votes reported by every polling station to see if they align with votes compiled with municipalities. If there are issues, a separate CSV file will be produced listing mismatches. Even if mismatches are found, the polling stations' reported totals will still be used to cross-check the values. This is because, during testing, some undefined formatting issues were found in the EML data, causing discrepancies. For example, some polling stations list mail-in votes separately from the station that processed them; others list them together.

Unfortunately, the EML files do not always have complete data. For example, in the House of Representatives election 2023, some municipalities failed to send the Kiesraad their collection form in time, resulting in uncorrected data being published in some of the municipality totals. This is not ideal, so centrally reported totals are not used; instead, the polling station reports are used.

4.2 Downloading Documents

There is no easy way to find all relevant documents that need to be downloaded. A list had to be manually compiled based on the *verkiezingen-processen-verbaal*[6] repository. Many files had to be manually checked to determine whether they should be included in Verelect's data. Unfortunately, this list is incomplete, as the original repository contains bad links; many are inaccessible, as municipalities usually host this data only for a short time after the election.

Several formatting issues exist among the files that are still available. Some files have invalid naming formats that require renaming to be recognized as PDF files. Some PDF files have malformed index table entries that cause problems when the file is opened with a less advanced PDF editor or library that cannot automatically repair these issues. This generally results in these files failing to be recognized or loaded by PDF tools. Sometimes, files get loaded, but every page becomes a blank 69-byte image. The workaround for these issues that had to be implemented is a list of common errors compiled and automatically corrected through a filter. Some more advanced prebuilt PDF libraries attempt to fix the broken index table, but they are not always successful.

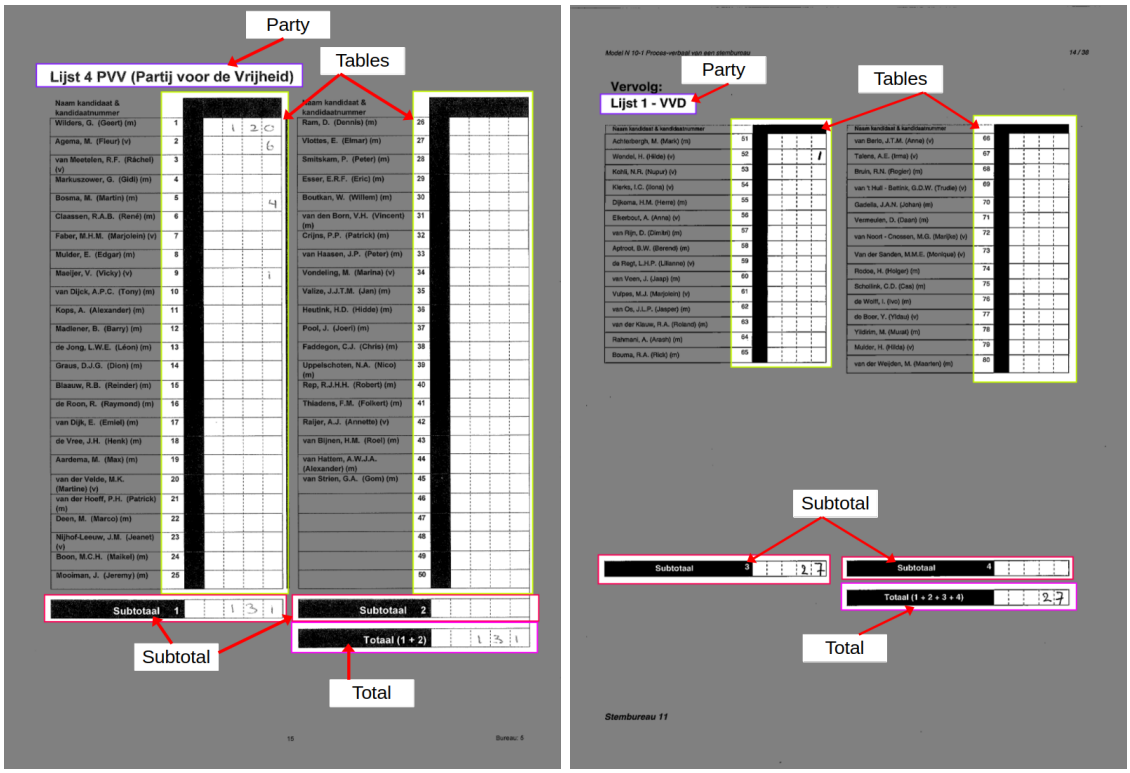
Another issue came from figuring out which polling stations generated which documents. Since all the files are hosted together, there is no consistent way to tell which polling stations released which data. Most municipalities create their own naming convention for the files they host. Most that follow this route usually put the polling station ID in the file name. So, every municipality was manually given a custom regex string to identify which polling stations this data belongs to. A few municipalities do not do this or, worse, have no naming convention, opting for filenames to be a randomly generated string of characters. For these municipalities, files had to be manually sorted by polling station. This would have to be a one-time task for each election, but it is still not ideal since it requires manual work.

Some municipalities released documents bundled together in large PDF files. For example, they compiled ten polling stations into a single document and then published that. This type of data is excluded as it is hard to tell which polling stations were included and which polling station each section belongs to. One municipality also wrapped its concatenated 500-page forms in one big zip file. If nonstandard behavior like this is encountered or the form is unavailable, it is marked as "Not downloaded" in the final results. This means they are automatically counted as missmatching, and no further processing or OCR steps will be taken.

The previously mentioned issue can also be seen in the publishing of correction forms. There is no consistent way that these documents are published. Some correction forms are prepended onto the previous document, some are published on their own, some are published instead of their original document, some only include the corrected data, and others include all of the original data with corrections. Due to these discrepancies, correction forms were excluded from the dataset. This results in mismatches because the processed forms have outdated and uncorrected data.

4.3 Finding Document structure

Unfortunately, no dataset was available to train the Form Element Model. Therefore, our first contribution is a custom-made dataset that is published publicly [7]. This dataset labels relevant document elements: party names, tables, subtotals, and totals. Two of these labeled forms can be seen in Figure 4.1. A common feature encountered while making the dataset was that many of the municipalities used vastly different styles of forms. So, the dataset had to account for as many variations as possible. Ideally, the data in the dataset must represent the real-life distribution of data to achieve the best possible results. The dataset was created from a random sample from the original data to account for this. In total, the dataset contains almost 1000 images of forms alongside a few images of irrelevant documents so that the model would avoid making predictions on other pages that do not contain the data it is looking for.



(a) (b)

Figure 4.1: Examples of 2 different labeled forms.

The model trained on this dataset is designed to categorize document pages as "relevant" or "irrelevant". A page is considered "relevant" if the model detects a party name, at least one table, and at least one total or subtotal. Otherwise, a page is marked "irrelevant"; it is discarded, and no further action is taken. When all available pages are parsed, the model compiles a list of all the forms and cross-references with data that was expected to be there based on the EML. That way, entries not found are marked as "Party page missing or parsed incorrectly".

Another issue the dataset accounts for is user error. Most commonly, when volunteers make errors, they cross out the original numbers and write new ones on the side. The dataset includes a number of these examples to allow the Form Element model to capture these corrections, even if they are out of bounds for the element itself. An example of how typos are compensated for in labeling can be seen in Figure 4.2.

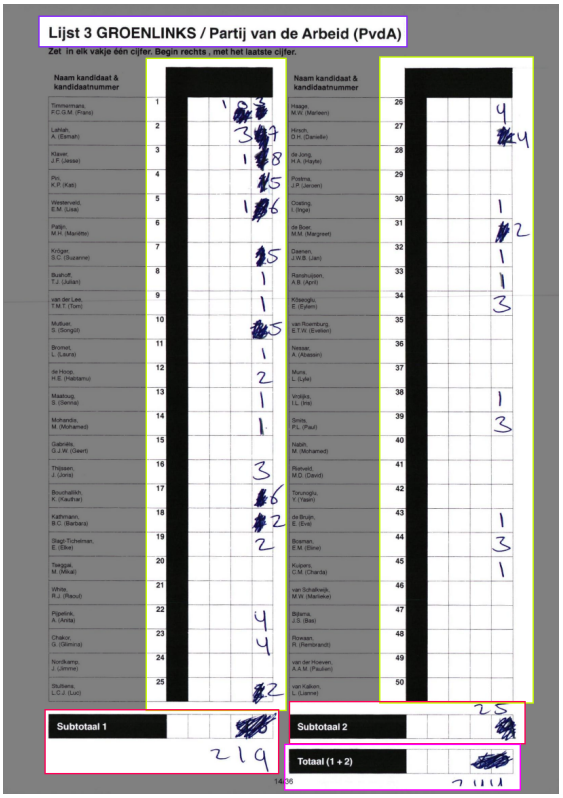


Figure 4.2: Example of form labeling when typos are present, and certain data is cut off by scanning.

A big issue encountered was that the model sometimes struggles to separate relevant text from irrelevant text next to it. This is a problem when detecting party names since, in some styles of forms, the party names do not have a consistent location or style, making

them easily confused with other text. More forms of this type were included in the dataset to account for this.

Some forms were malformed to begin with. While scanning, mistakes can lead to forms losing valuable information, making the forms unprocessable. Unfortunately, this is a systemic issue, and nothing can be done about this from the software side. One of these forms can be seen in Figure 4.3.

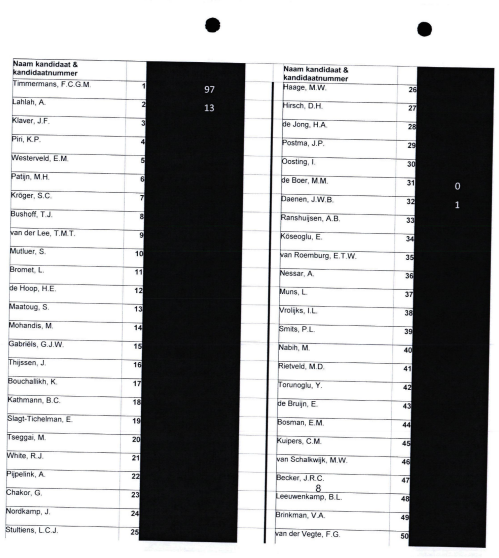
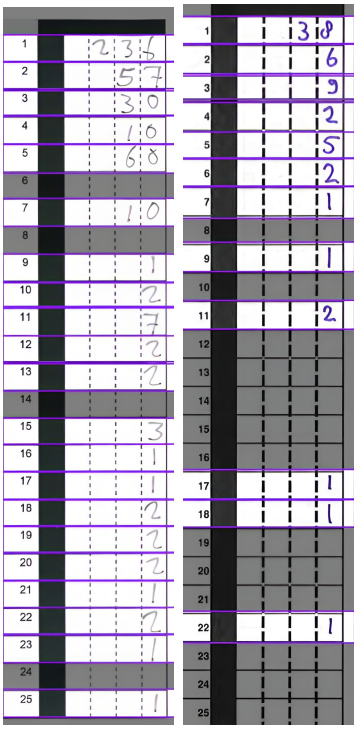


Figure 4.3: Example of a malformed form where needed information like party name is missing.

4.4 Filtering rows

Our second contribution is the custom dataset for the Data Row Model. It is also publicly available [8]. The dataset is made out of a random sample processed by the Form Element Model, and then the resulting detected tables are added to the dataset. This dataset aims to detect and extract rows with data from the tables detected. Two example labeled tables can be seen in Figure 4.4. There are two main reasons this is necessary. Firstly, rows the model does not detect are assumed to be empty. Empty rows get marked as 0 in the results, which cuts down the processing time and cost required by the OCR because most rows in the data are empty. Secondly, the model allows for further preprocessing steps before OCR is applied. The primary use of the preprocessing is to introduce more blank space between rows, increasing OCR performance.

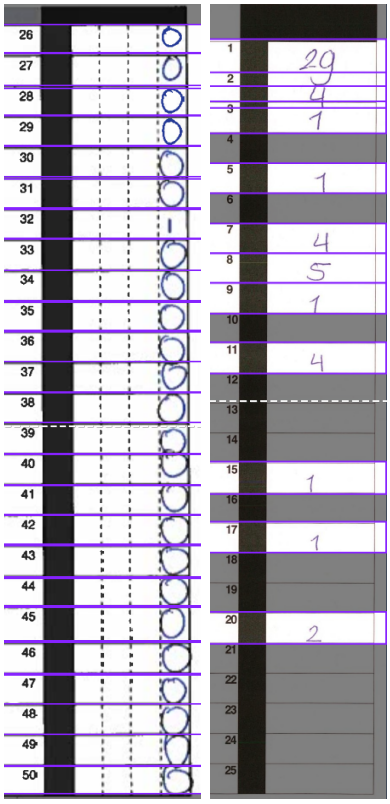


(a) (b)

Figure 4.4: Examples of 2 different labeled tables. Each row with data gets added to the dataset.

Volunteers usually do not have perfect handwriting, so some steps had to be taken to compensate for this. Instead of sticking with the table's row borders, this model aims to include the entire character in its segmentation, even if it extends beyond the boundaries. This is present in the dataset, as many labels intersect with one another, which reflects the messiness of the handwriting.

A significant issue encountered when training this model is that volunteers sometimes write numbers big enough to overlap multiple rows. Those numbers can sometimes take up two or more boxes vertically, making the model identify them as two rows with data. The dataset attempts to remedy this by adding more difficult examples. Not all of the issues can be remedied depending on how big the writing errors are. Another issue occurs when some volunteers decide they cannot leave boxes empty, so they attempt to fill every row with miscellaneous characters to signify that they are empty. When this occurs, the model marks every row as having data, leading to unnecessary data being processed by the OCR. Examples of both of these issues can be found in Figure 4.5.



(a) (b)

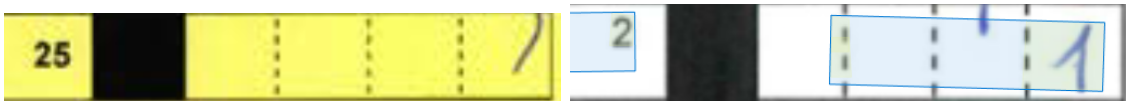
Figure 4.5: (a) Shows labeling with empty rows filled with zeroes. (b) shows labeling when multiple rows overlap.

4.5 The OCR

The OCR is the final step of parsing election forms. It takes its inputs from both models. First, it detects which party the page belongs to. Then, it proceeds to read all of the detected rows. The names of the tables are skipped. This is because identifying them based on their ID is more convenient and accurate. The candidate's vote total is read and compared to the one found in the EML data and the index number. The detected characters are added to the results CSV, and the status is appropriately given as "Matches" or "Mismatch".

One issue this system has is the problem, as mentioned earlier, of marking empty rows. When these markings are taken away from their context, they can start looking like digits; this is a big problem in Groningen forms. Figure 4.6(a) shows an example of this. Another issue encountered is with the styling of forms. The model often interprets the styling of

some table elements, like borders, as digits. This is especially common when table borders are made of dashed lines in black font. These elements often get interpreted as colons or the digit one. 4.6(b) is an example of table styling being interpreted as elements. OCR models can be influenced in a few ways to improve performance. One great way to do it is through preprocessing. Some unwanted table elements can be removed from the image through erosion and dilation. Increasing padding between rows also significantly increased performance since most models are trained to detect smaller characters on an image instead of ones spanning the entire image.



(a) (b)

Figure 4.6: (a) Shows out-of-context marking looking like numbers, in this case, a 1. (b) showcases table borders getting interpreted as a number. In this case, four ones are being detected instead of one.

5

Implementation

Verelect is implemented entirely using Python. It uses eml2sql [4] to parse eml files into an SQLite format. After this step, the SQLite data is queried to be compiled into an easy-to-use CSV where unnecessary data like voter turnout is discarded. Verelect heavily uses the file system on which it is being hosted. Every intermittent step is saved on the filesystem to recover easily during an error. This is important since some modules can take multiple days to finish running.

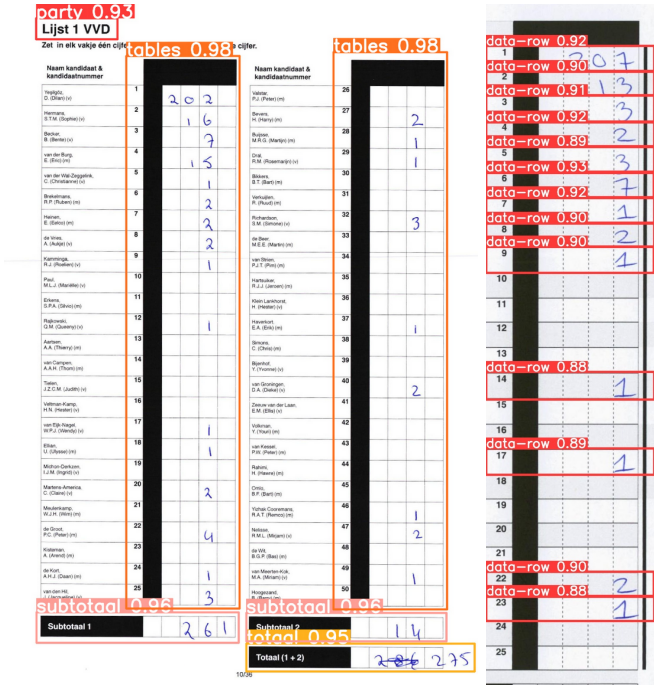
5.1 Models

The Form Element Model and the Data Row Model datasets were created using Roboflow [9]. The models were trained based on the YOLOv9e[10] pre-trained model. They were trained for 300 epochs with a file size of 640 pixels. An example of a detection can be seen in Figure 5.1.

5.2 OCR

For reading party names and differentiating between totals and subtotals, the Tesseract OCR engine [11] is used. It performs well when reading printed text on a page. Tesseract runs locally in the host machine, allowing for reliable and cost-effective recognition. Managing a local instance avoids issues if the system becomes deprecated or unavailable. Its main downside is that Tesseract cannot recognize handwritten text needed for Verelect.

Tested Models Kraken [12] and PaddleOCR [13] could recognize handwritten text out of the box but were often confused by table elements like rows and columns, with their results frequently being thrown off. Figure 5.2 shows an example of this behavior. Custom



(a) (b)

Figure 5.1: (a) An example of form elements being detected alongside confidence levels. (b) An example of rows with data being detected alongside confidence levels.

OCR models were also trained based on the MNIST [14] and HWD+ [15]. Alongside many preprocessing steps, they yielded excellent results when the handwriting was clean, with the downside that it could not cope with the character intersecting a table line. This feature occurs very frequently in the data, so the OCR model chosen has to be able to work with it.

Figure 5.2: Paddle OCR engine being thrown off by table styling. Interpreting elements as actual text. Results are shown on the right alongside their confidence levels.

Final Choice In the end, we settled on using Microsoft Azure Document Intelligence [16]. It is a proprietary document analysis tool that includes OCR. While it still suffers from

the same issue that Kraken and PaddleOCR do, it does so to a significantly lesser degree. The downside of using this model is that it cannot run locally and requires payment. Without any optimizations, it would cost around €329 to parse data for the House of Representatives election of 2023. The cost is estimated based on how many rows with data were detected, assuming that the price per thousand pages is €1.388. This is taking into account that, as explained in 4.2, much of the data is missing. Realistically, on a complete dataset, the operation would cost somewhere around €450.

A batching system is implemented to reduce this cost. Rows that are detected are grouped based on their party affiliation and appended together to a single page. Depending on how many rows of data were detected, multiple groups could be appended onto a single page. The group's location on the page is remembered and passed into the Microsoft API. After the results are returned, they are split based on the original layout of the groups. This method significantly reduced the OCR cost to around €69 for the before-mentioned election. The performance also increases generally since most models are optimized for reading text blocks with static gaps between them. This has a side effect that benefits the OCR false positives. Since rows are attached with more padding than they appear in the tables, the OCR engine does not make mistakes when it interprets rows close together as a single block of vertical text. An example of images generated by this batching system can be seen in Figure 5.3

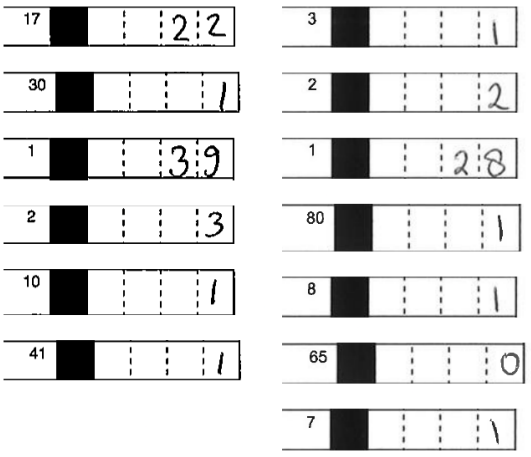


Figure 5.3: Batched row images generated by the batching system.

6

Evaluation

The Verelect system was tested on an Ubuntu 22.04 LTS server running an Intel i7-6700K CPU and an NVIDIA Geforce RTX 3050 graphics card.

6.1 Acquiring the dataset

67.7% of all forms listed in the dataset were successfully downloaded and categorized into a proper dataset. 75 municipalities had no data available, meaning verifying their results was impossible. The most significant contributing factor to this number was that many forms were unavailable. This statistic also includes documents that were included in the eml dataset but were never added to the *verkiezingen-processen-verbaal*[6] repository that Verelects list is based on, out of the 267 municipalities included in the dataset for Verelect. Eighty-seven required some manual editing to get them into a usable format. This implies that Verelect does not support data that is out of the box for one-third of the municipalities.

6.2 Document Structure

The model is not evaluated by the number of correct detections but by how many pages it can fully detect without making mistakes that would lessen the quality of the data extracted. This is used because, for this thesis, a single correct detection in a page of errors yields no usable data. More accurately, a page is considered detected if a party name, at least one table, and a total or a subtotal have been extracted.

While testing on a relatively small sample, 6654 out of the 6833 pages (97.38%) were parsed correctly. This sample was more sanitized than a real-world dataset, as malformed images were filtered out. This mainly consists of scanning errors where the page was not

entirely scanned, resulting in missing crucial data like vote totals or party names. From the available data on the 2023 election, 161972 pages out of 170399 (95.05%) succeeded in parsing correctly.

6.3 Row Recognition and OCR

For simplicity, the evaluation metric for the Data Row Model is the same as that for the previously mentioned model. However, without significantly expanding the test dataset, it is impossible to accurately and automatically measure how many rows have been detected correctly, and this requires extensive manual work. To evaluate the model, it was necessary to inspect each table manually to see if it was detected correctly. Due to the amount of manual work it would take to evaluate each table manually, this model's performance on the 2023 House of Representatives election is evaluated together with the OCR engines.

In a randomly taken sample of 1935 tables from all municipalities, the model made at least one mistake in 35 tables. Like the previous one, this sample has the same caveat: it is a controlled environment where tables are filtered for malformations.

Combining the OCR model and the Data Row Model together, out of the 6449852 rows, 6091758 (94.45%) were extracted and identified correctly, meaning that only one in every twenty rows needs to be checked by a human. However, this number does not represent the actual performance of the models since most rows are empty. When the Data Row Model does not capture a row, it automatically captures it as "0". Because of this, the accuracy is greatly inflated. Filtering for empty rows, 257193 out of 615287 (41.8%) were identified correctly. These results can be better visualized by grouping rows by the pages in which they appear. Of all correctly parsed pages, 64359 (39.73%) had every row correctly identified. This means human reviewers could skip all those pages entirely, as they do not have a single discrepancy detected.

7

Discussion

Overall, Verelect successfully reduced the workload required to verify the Dutch election. It is rather hard to say how much time it would save potential verifiers because of the human factor. Each person has their own workflow and will approach the task and use this tool differently.

Performance Verelect is particularly effective in verifying candidates that received no votes. While this may seem insignificant, it still saves potential verifiers a lot of time and effort going through blank pages and checking if they are indeed blank. This effectiveness is amplified by the fact that most rows in the election data are blank, resulting in potentially considerable productivity increases for users. This is especially apparent in smaller municipalities where most pages are blank, resulting in lots of pages that can be skipped entirely. Verelect performs worse on rows with data, but it is not useless. Because its performance is heavily dependent on the form's style and the quality of the volunteer's handwriting, it is likely to correctly verify entire pages and whole documents at once. Depending on which polling station the verifier desires to verify, this might entirely eliminate the need for human interaction.

Availability The form downloader part of Verelect has problems. Since some data is unavailable for those municipalities, Verelect will not provide any benefits. The amount of unavailable data is significant as, currently, Verelect cannot function in around one-third of the municipalities in the Netherlands. In some municipalities, Verelect might hinder verification more than it helps.

The way some municipalities host data is generally the weakest point of the entire Verelect system. Since some municipalities publish files in ways that are not supported by

Verelect, the work required to fix these documents might be more than the time saved by automatic verification assistance. This is exacerbated by the corrigendum issue since Verelect is still incapable of reading those documents, leading to false positives appearing in the output even if no errors occurred.

8

Limitations

Form Issues The Form Element model often has inconsistent Party name detection. It was common for other irrelevant text to be detected as the party name, and the actual party name would be ignored. This occurred because some forms have the party names right next to many other text, confusing the model. Furthermore, the party name has no set location and style, making detection difficult on more uncommon forms with a lower presence in the dataset.

Table Issues Sometimes, the Data Row Model fails to align with tables of uncommon sizes. If the table being analyzed is smaller than expected, the model might fail to align its detections with the table's size, resulting in the wrong parts of the page being extracted. This causes problems at the OCR step where missing numbers will cause mismatches. Another issue relates to the aforementioned occurrence where volunteers would choose some random marking to indicate an empty row. This causes problems since the Data Row Model is more likely to detect an empty row accurately than the OCR module. When these markings occur, the Data Row Model cannot detect them as empty and must pass them to the OCR module, making accuracy worse. This model also has an issue where it fails to deal with changes in rotation correctly. This is because documents with rotated elements are rare and heavily underrepresented in the dataset.

OCR Issues The form's style is a major contributing factor to the OCR model's performance. Not all forms are made equally, as spacing, border style, color, and handwriting play a significant role in the performance of the OCR. Forms that contain dotted or dashed columns and row borders are major contributors to errors, as the OCR keeps picking them up as ones, semicolons, colons, dots, and other punctuation. A similar issue occurs when

writing errors are made on a form. OCR engines, in general, are designed to detect extracted text, but they struggle to distinguish between crossed-out text and normal text.

An Impractical Solution All of the problems discussed in this section can be solved by making bigger and more detailed datasets. Both the Form Element Models and the Data Row Model suffer from datasets that are too small and do not include many weird quirks of each municipality. The OCR performance can be optimized by creating a custom dataset containing many examples from the previous elections. In this case, we would need a highly detailed dataset of digits filled into every table style that could be used in an election. While these additions would certainly improve the performance, implementing these changes would require a lot of manual work gathering data and labeling it. Data gathered would need to be unique and personalized to each individual municipality. If, after this dataset is created, one municipality decides to change something, it would be necessary to rebuild its dataset to better suit them again. This amount of manual work making datasets would require an unreasonable amount of effort and is not realistic. A more fitting approach would be to systematically change the election system to be more compatible and convenient for tools like Verelect.

9

Design Recommendations

9.1 The Importance of Consistency

A current issue is that each municipality has differences in how it publishes data, what style of forms it uses, and how it handles missing values and corrigendums. When used with machine learning methods, this introduces a lot of variance into the models, which makes results noticeably worse. Even if the other proposed changes were implemented but not made consistent across every municipality, they would have little to no effect. This is because it would be a lot of work to implement two different systems, one for municipalities that adopt them and one for the others.

Differences between municipalities must be minimized to solve these issues. All municipalities should adopt the same standard of form and agree on dealing with issues like marking empty rows and publishing corrigendums. Even if none of the other two proposals were implemented, this simple change would significantly increase the effectiveness of OCR and other machine learning models. It would also allow for more automation in the election process, making it easier to work with this data.

9.2 Centralized Document Publishing

The first problem that needs to be tackled is the availability of data. For data to be easily accessible, it should be:

- Data needs to be hosted fully; verification is useless when you can only verify two-thirds of the votes, with the others being a black box.
- Data needs to be published systematically; there is no use for data that cannot be found.

- Data needs to be hosted in a consistent and clean format with its complete metadata. Documents published should not require a preprocessing step just to be opened.

This thesis proposes a centralized document hosting system to address these issues. The main idea is that the Kiesraad, the organization responsible for overseeing the election, should handle the hosting instead of requiring each municipality to host its own data. Under this system, each municipality would fill out a form with the necessary metadata. Figure 9.1 shows an example of such a form. This system also needs a user and machine-friendly interface to browse and navigate the forms. Such a system would eliminate the need for all the workarounds used for Verelect's data acquisition.

A centralized system has the downside of introducing a single point of failure. A single fault in this system could affect the transparency of the entire system. Arguably, this is not a major problem due to the nature of the election. Even a few forms becoming unavailable or compromised under the current system could introduce doubt and uncertainty for the organizations and government involved. This issue could also be compensated by being implemented alongside the current one. In this case, you could get the benefits of centralized hosting without the single point of failure problem.

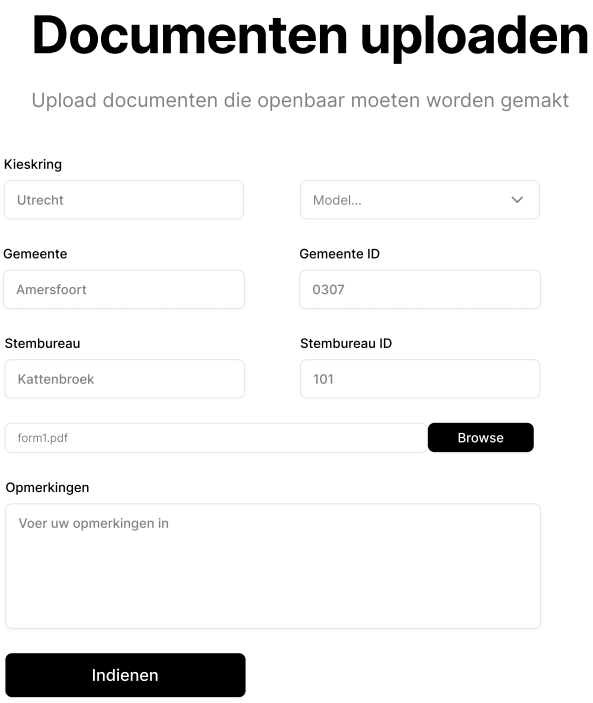
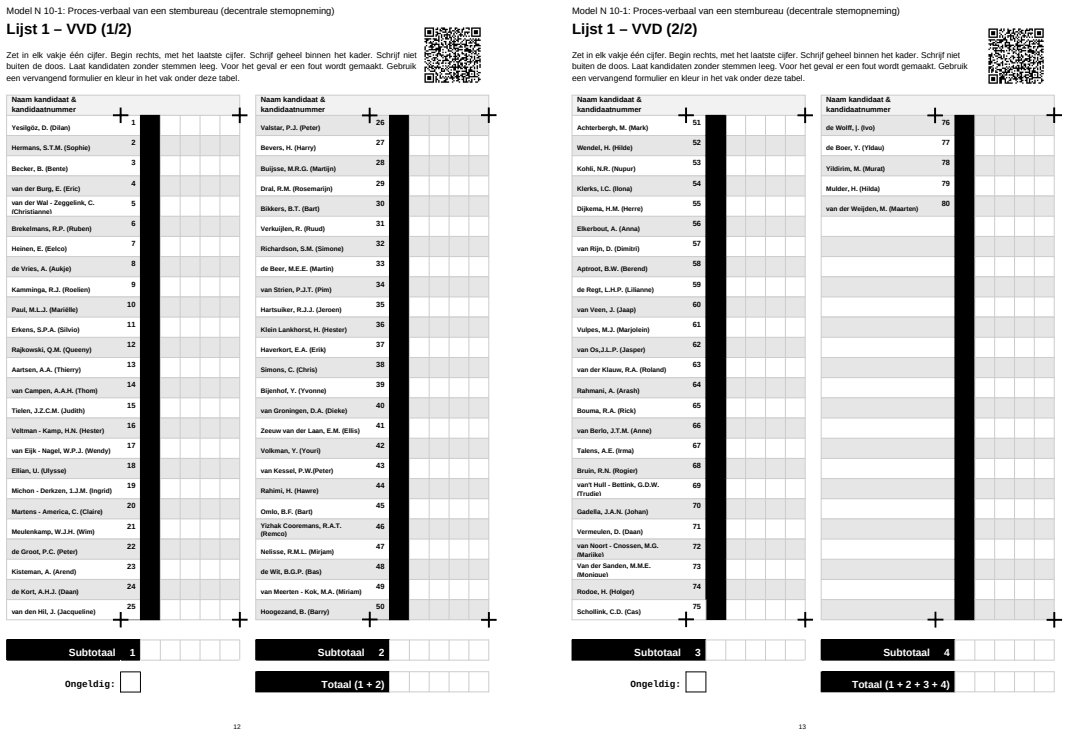


Figure 9.1: Potential upload design of the centralized document hosting system

9.3 Proposed Template

The second issue that needs to be solved is the form design. Forms should be designed to accommodate not only human readers but also machines. The current form design does not attempt to make them accessible for parsing and OCR. To solve this, this thesis proposes a modified form template intended to optimize OCR and the performance of other machine learning methods. This new form can be seen in figure 9.2.



(a) (b)

Figure 9.2: Template of the proposed form style. (a) shows the standard total page, (b) shows a continuation of there are more than 50 candidates

When parsing pages, determining where the page comes from is often challenging. This new template features a QR code at the top right of the page. This QR code encodes up to 406 bytes of data into a data format format designed for machines to read. This is enough to encode simple JSON strings containing data about the page. The code in the example encodes municipality ID, polling station ID, party ID, page number, total pages of documents, total candidates of the current party, candidates on the current page, form

number, which candidate ID starts the page, and which candidate id is last on the page. This feature would solve the Form Element Models issue with locating party names since it would always be encoded in the QR code, which is far easier to detect.

A general issue with forms is corrections made by volunteers. They cause the OCR to miss-detect crossed-off numbers as undefined characters, marking them as mistakes. To solve this issue, a check box is added at the bottom of the page, intended to be marked once a form needs to be corrected. Once it is marked, a new form is printed, and the current one is discarded. The checkbox is intended to signify that the form is obsolete and should not be used in the final result. This would allow the polling stations to print more of these forms, removing the need for scribbling out errors. This will hopefully improve the handwriting quality of the form, increasing OCR reliability.

The most crucial change in the forms is the newly redesigned tables. Table corner markers are added to help machine learning methods more easily identify table boundaries. Tables are also capped at a constant size, which benefits the Data Row Models' ability to identify the row boundaries correctly. Table borders are set to a lighter color than normal. This is highly beneficial because table borders can be removed by grayscaling and thresholding. This allows any pen writing to be easily isolated from table elements, allowing easier detection and recognition. This effect can be seen in Figure 9.3.

(a) (b)

Figure 9.3: (a) Row in the proposed form template filled in. (b) Filled in a row after greyscale and thresholding preprocessing steps were applied.

9.4 Potential Outcome

If these proposed changes were implemented, they would greatly improve Verelect's performance in almost every way. The Form Element Model and the Data Row Model could achieve far greater accuracy since the major limiting factor of inconsistent styling would be eliminated. The OCR would be much more accurate even with messy documents because of more effective preprocessing options. This would result in the system parsing many more pages correctly, reducing humans' workload.

More importantly, this would solve the document acquisition problems. Since forms would now be centrally hosted, their availability and distribution would be standardized, allowing Verelect access to all election documents and the ability to work with corrigendum forms. This could potentially even make the system more useful as a reliable verification tool instead of just a first filter.

10

Related Work

This thesis was created as a "continuation" of a previous thesis written on this subject. [17] That thesis described a design for an OCR-based tool for verifying elections like Verelect. But unlike Verelect, the system was purely OCR-based; it only read the raw numbers and could not detect which party and candidate these votes belonged to. TensorFlow OCR was used as the OCR engine of choice. No statistics or indication of accuracy have been released, so it is unknown how this system compares to Verelect's performance. Some issues regarding published data, like document scanning errors, were briefly mentioned. However, most other problems mentioned in this thesis were ignored and not touched upon.

Not much research has been done on this topic, but table recognition has been explored in other use cases. T. Kazdar et al., in their paper "Table Recognition in Scanned Documents"[18], presented a partial workflow to use deep learning to parse tables in invoices automatically. Their approach to detecting tables and their structure was slightly different. Instead of detecting visual features of tables, they used a Tesseracts OCR engine to capture text in a document and, out of that, detect table bounds and rows and columns. Their primary benefit is that invoices have no handwritten text, unlike the election forms. This allows "off-the-shelf" OCR engines to be more consistent.

Similar Elections Similarly to the Netherlands, Romania also holds its elections entirely on paper ballots and forms. Since 2019, they have been using their own SIMPV system. [19] One crucial facet of this system is that it ensures the transparency of the election by providing incremental updates of the election process live as they are occurring to the general public. It shares some similarities with the proposals shared in this thesis.

Election data is published centrally on the government's website. The data published is extensive and complete. All of the data from the past 18 elections since this system was

founded is fully available. This system not only hosts fully filled-in and signed versions of election documents but also takes snapshots of these documents as they are created and published. The documents are tied based on which level of government issued them, which allows for the election to be looked at more or less granularly. The documents are provided with a transcript, allowing users to crosscheck the contents more easily. The physical documents also contain QR codes, allowing easier identification of their origin and purpose.

11

Conclusion

This thesis analyzed the potential of a machine learning based verification system for Dutch election results. It was demonstrated that a significant part of the election results could be accounted for automatically. The Form Element Model and the Data Row Model are capable of parsing the election vote total form structure, which is an important step when trying to extract data from a document. The datasets for these models are released publicly to allow for easier creation of such tools as verelect.

Verelects' performance could have been improved a lot, but a number of limiting design problems were discovered with how the Dutch electoral system functions. These problems span a lot deeper than what is possible to remedy with software, so potential systematic changes were proposed that could be enacted to make future research involving election results and manually scanned forms possible. These solutions include publishing, styling, and organizing these forms between different municipalities.

References

[1] TadMaj. **Verelect**. https://github.com/TadMaj/verelect, 2024. 1

[2] The Ministry of General Affairs. **Models | Election of the House of Representatives**. 3

[3] Kiesraad. **EML-standaard**, 9 2020. 5

[4] B. Hubert, C. Mostert, M. Janssen, and H. Bouwkamp. **eml2sql**. https://github.com/kiesraad/eml2sql, 2024. 5, 19

[5] Khurram Azeem Hashmi, Marcus Liwicki, Didier Stricker, Muhammad Adnan Afzal, Muhammad Ahtsham Afzal, and Muhammad Zeshan Afzal. **Current Status and Performance Analysis of Table Recognition in Document Images With Deep Neural Networks**. *IEEE Access*, **9**:87663–87685, 2021. 5

[6] S. Provoost. **verkiezingen-processen-verbaal**. https://github.com/Sjors/verkiezingen-processen-verbaal, 2024. 11, 23

[7] Dutch Election Forms. **Form-Element-Recognition Dataset**. https://universe.roboflow.com/dutch-election-forms/form-element-recognition-uppvt, apr 2024. visited on 2024-05-19. 13

[8] Dutch Election Forms. **Form-Data-Rows Dataset**. https://universe.roboflow.com/dutch-election-forms/form-data-rows, apr 2024. visited on 2024-05-19. 15

[9] B. Dwyer and J. Nelson. **Roboflow (Version 1.0)**, 2022. 19

[10] Chien-Yao Wang and Hong-Yuan Mark Liao. **YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information**. 2024. 19

REFERENCES

[11] S. Weil. **tesseract**. https://github.com/tesseract-ocr/tesseract, 2024. 19

[12] Benjamin Kiessling. **kraken; kraken documentation — kraken.re**. https://kraken.re/main/index.html, 2015. [Accessed 16-06-2024]. 19

[13] Yuning Du, Chenxia Li, Ruoyu Guo, Xiaoting Yin, Weiwei Liu, Jun Zhou, Yifan Bai, Zilin Yu, Yehua Yang, Qingqing Dang, and Haoshuang Wang. **PP-OCR: A Practical Ultra Lightweight OCR System**, 2020. 19

[14] Li Deng. **The mnist database of handwritten digit images for machine learning research**. *IEEE Signal Processing Magazine*, **29**[6]:141–142, 2012. 20

[15] Cédric Beaulac and Jeffrey S. Rosenthal. **Introducing a New High-Resolution Handwritten Digits Data Set with Writer Characteristics**. *SN Computer Science*, **4**[1]:66, November 2022. 20

[16] Lei Cui, Yiheng Xu, Tengchao Lv, and Furu Wei. **Document AI: Benchmarks, Models and Applications**. November 2021. 20

[17] R. Wijnbergen. **Stage Verslag**. Technical report, Leiden University of Applied Sciences, 2023. 35

[18] Takwa Kazdar, Marwa Jmal, Wided Souidene, and Rabah Attia. **Table Recognition in Scanned Documents**. In Ngoc Thanh Nguyen, Yannis Manolopoulos, Richard Chbeir, Adrianna Kozierkiewicz, and Bogdan Trawiński, editors, *Computational Collective Intelligence*, pages 744–754, Cham, 2022. Springer International Publishing. 35

[19] AUTORITATEA ELECTORALĂ PERMANENTĂ. **NORME METODOLOGICE din 9 octombrie 2019**, 2019. https://legislatie.just.ro/Public/DetaliiDocumentAfis/218806. 35