University of Europe for Applied Sciences

Master Thesis

Master of Science (M.sc)

Department of Tech and Software

Major: Data Science (DS)

Topic: Automated Administrative Document Processing

Author: Arkadii Shekhovtsov

Matrikel-Number: 61191937

First supervisor: Prof. Dr. Raja Hashim Ali

Second supervisor: Prof. Dr. Talha Ali Khan

University of Applied Sciences Europe
Iserlohn · Berlin · Hamburg

EIGENSTÄNDIGKEITSERKLÄRUNG / STATEMENT OF AUTHORSHIP

Arkadii	Shekhovtsov
Name | Family Name	Vorname | First Name
61191937	Automated Administrative Document Processing
Matrikelnummer | Student ID Number	Titel der Examsarbeit | Title of Thesis

Ich versichere durch meine Unterschrift, dass ich die hier vorgelegte Arbeit selbstständig verfasst habe. Ich habe mich dazu keiner anderen als der im Anhang verzeichneten Quellen und Hilfsmittel, insbesondere keiner nicht genannten Onlinequellen, bedient. Alles aus den benutzten Quellen wörtlich oder sinngemäß übernommen Teile (gleich ob Textstellen, bildliche Darstellungen usw.) sind als solche einzeln kenntlich gemacht.

Die vorliegende Arbeit ist bislang keiner anderen Prüfungsbeh.rde vorgelegt worden. Sie war weder in gleicher noch in ähnlicher Weise Bestandteil einer Prüfungsleistung im bisherigen Studienverlauf und ist auch noch nicht publiziert. Die als Druckschrift eingereichte Fassung der Arbeit ist in allen Teilen identisch mit der zeitgleich auf einem elektronischen Speichermedium eingereichten Fassung.

With my signature, I confirm to be the sole author of the thesis presented. Where the work of others has been consulted, this is duly acknowledged in the thesis' bibliography. All verbatim or referential use of the sources named in the bibliography has been specifically indicated in the text.

The thesis at hand has not been presented to another examination board. It has not been part of an assignment over my course of studies and has not been published. The paper version of this thesis is identical to the digital version handed in.

05/08/2025 Potsdam	[illegible]
Datum, Ort | Date, Place	Unterschrift | Signature

University of Europe for Applied Sciences

Declaration on the use of generative Artificial Intelligence (AI) systems

Shekhovtsov	Arkadii
Name | Family Name	Vorname | First Name
61191937	Master thesis
Matrikelnummer | Student ID Number	Titel Prüfungsarbeit | Title of the exam

I have used the following artificial intelligence (AI)-based tools in the creation of my work:

1. ChatGPT (free Version 4o, OpenAI);
2. Writefull for Overleaf.

I further declare that

☐ I have actively informed myself about the performance and limitations of the above-mentioned AI tools,
☐ I have marked the passages taken from the above-mentioned AI tools,
☐ I have checked that the content generated with the help of the above-mentioned AI tools and adopted by me is actually accurate,
☐ I am aware that, as the author of this work, I am responsible for the information and statements made in it.

I have used the AI-based tools mentioned above as shown in the table below.

AI-based support tool	Usage	Parts of the work affected	Remarks
ChatGPT (free Version 4o), OpenAI	• Rephrasing text • Code assistance	• Theoretical Background • Visualization of model outputs	• Used to improve clarity and structure • Used to overlay detection results on input images
Writefull	Grammar correction	Throughout document	Assisted in grammar correction

Potsdam, 05.08.2025

[illegible]

Abstract

In this thesis, I present an automated pipeline for processing administrative documents with a particular focus on receipts containing manual checkmarks. The motivation stems from real-world scenarios such as expense tracking and reimbursement, where marked items on receipts indicate exclusions or personal purchases. My system aims to transform such annotated paper receipts into structured digital tables with high accuracy.
I used modular architecture combining traditional and deep learning-based OCR methods (Tesseract, EasyOCR) and checkmark detection via object detection algorithms (YOLOv8). A hybrid dataset was developed, consisting of handmade and synthetically generated checkmark images annotated in YOLO and COCO formats. I evaluated multiple object detectors, including YOLOv8n, Faster R-CNN, and SSD300, on this dataset.

The proposed pipeline was tested on a variety of real-world receipts with varying complexity, and integrates OCR, checkmark localization, item-price matching, and validation against user-provided totals. My experiments demonstrated that YOLOv8n achieves a mean Average Precision (mAP@0.5) of 0.995 on checkmark detection, outperforming all alternatives. EasyOCR and Tesseract were used for text recognition, each compensating for the other's weaknesses.

The full pipeline succeeded in extracting accurate structured data from 57% of the test receipts, with a clear path to improvements. Identified failure modes include OCR misclassification, checkmark detection failures, and structural misalignment between text and marks. Future work proposes resolution enhancements, fallback OCR engines, improved preprocessing strategies, and potential use of LLMs for ambiguous parsing.

This study contributes a reproducible, open-source framework for intelligent receipt understanding and introduces novel methods for integrating OCR and object detection for real-world administrative tasks.

Table of contents

List of figures

List of tables

List of listings

List of Abbreviations

OCR	Optical Character Recognition
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
CRNN	Convolutional Recurrent Neural Network
CTC	Connectionist Temporal Classification
mAP	mean Average Precision
FPS	Frames Per Second
SSD	Single Shot MultiBox Detector
R-CNN	Region-based Convolutional Neural Network
YOLO	You Only Look Once
FPN	Feature Pyramid Network
API	Application Programming Interface
GPU	Graphics Processing Unit
CPU	Central Processing Unit
PCA	Principal Component Analysis
COCO	Common Objects in Context
JSON	JavaScript Object Notation
LLM	Large Language Model
PSM	Page Segmentation Mode
HMM	Hidden Markov Model
DTW	Dynamic Time Warping
EM	Expectation-Maximization
SOM	Self-Organizing Map

1. Introduction

In everyday life people often need to deal with paper receipts. This could be tracking personal expenses, splitting a bill with friends at a restaurant or fulfilling business trip reports for an employer. Typically, processing such information requires manual data entry, which is inconvenient and sometimes time consuming.

Take a photo of the receipt and get structured data, that’s what people want. Moreover, it will be useful to have an opportunity to mark which items should be taken into account. The easiest way is to put checkmarks next to the desired/undesired items directly on the paper receipt.

The task is to create an automated pipeline that allows:

1) Processing photographs of receipts with manual marks
2) Extracting text and mark information
3) Generating a final digital table with the required expenses

This approach is different from typical commercial solutions, which often rely on fixed templates. In contrast, this work aims to support flexible, user-driven selection of items, using informal handwritten marks like checkmarks. Technical implementation details, model comparisons and dataset preparation are described in the following chapters.

2. Literature Review

2.1. OCR general

2.1.1. Definition of OCR

Optical Character Recognition or Optical Character Reader (OCR) is an automated technology for images conversion of printed or handwritten text into machine-encoded text using computer algorithms. It includes both electronic and mechanical conversion, however the second method was relevant at the very beginning of OCR development.

OCR systems work by analyzing scanned documents or photographs, detecting text regions, segmenting individual characters or words, and recognizing them using various methods, including machine learning and deep learning. The resulting digital text can be edited, searched, and processed automatically. [1]

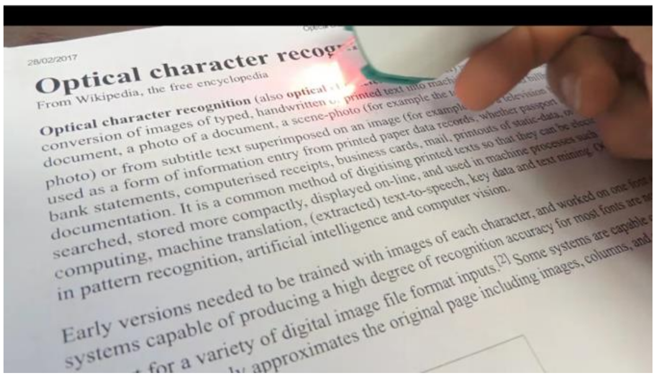


Figure 1. Scanning and optical character recognition (OCR) process in real time using a portable scanner

2.1.2. Brief History of OCR

Early optical character recognition can be traced to technologies developed for telegraphy and assistive devices for the blind. In 1914, Emanuel Goldberg invented a machine capable of reading printed characters and converting them into standard telegraph code. Around the same time, Edmund Fournier d'Albe introduced the Optophone, a handheld scanner that, when moved across a printed page, produced distinct tones corresponding to specific letters or characters - effectively allowing blind users to “hear” printed text.

In the 1920s–1930s some of the first mechanical systems for reading printed text were invented. Early devices relied on templates and photodetectors, and were limited to specific fonts or specially prepared documents.

By the 1950s, the first commercial OCR machine was developed. It could convert printed text to computer-readable form. In the following decades, special fonts like OCR-A and OCR-B were introduced to simplify machine reading, and national postal services in the US, UK, and Canada began using OCR for sorting mail and processing documents.

A significant breakthrough came in the 1970s with the development of “omni-font” OCR systems. As computing power increased, OCR software became commercialized and widely adopted for digitizing large volumes of printed material. In the 2000s, OCR moved to the cloud and mobile devices, enabling real-time text extraction and accessibility features for visually impaired users.

Modern OCR systems leverage neural networks and deep learning, offering high accuracy for both printed and handwritten text in diverse languages and layouts, and powering applications from document management to instant mobile translation. [2]

2.1.3. OCR types

Optical character recognition systems are usually divided into two categories: online and offline. The naming of “online” OCR can be misleading for the general user, as it refers not to internet-based processing but to real-time dynamic input, while “offline” deals with static images.

Table 1. OCR Methods Comparison

Method	Text Type	Phases	Recognition Algorithms
Offline OCR	Printed and Handwritten (static)	Preprocessing, segmentation, feature extraction, classification, postprocessing	Template matching, k-NN, CNN, RNN, SOM, EM, clustering
Online OCR	Handwritten (dynamic)	Temporal feature capture, dynamic analysis, classification	Direction-based algorithms, DTW, RNN, HMM

These categories differ not only in input type but also in applied methods: offline OCR focuses on image processing techniques, segmentation and static feature extraction, while online OCR leverages temporal features, direction-based algorithms, and dynamic time warping (DTW) for handwriting. [3]

2.2. Phases of offline OCR System

While the receipt processing needs only offline OCR, let's explore its main phases. They can be listed as: image acquisition, preprocessing, segmentation, feature extraction, classification, and postprocessing.

Table 2. Phases of OCR

Phase	**Component**	**Purpose**	**Example Techniques**
Image acquisition	Scanner / Camera	Capturing document image	Scanning, camera capture
Preprocessing	Preprocessing module	Enhance image quality	Enhance image quality
Segmentation	Segmentation module	Separate text lines, words, characters	Separate text lines, words, characters
Feature extraction	Feature extraction module	Identify distinguishing features	Zoning, projection, border transition, graph matching
Classification	Classifier	Assign character class	Template matching, k-NN, CNN, RNN, SOM, EM
Postprocessing	Postprocessing module	Improve Accuracy	Dictionaries, language models, n-grams

2.2.1. Image Acquisition

This step is about capturing the document image, usually with a scanner or digital camera. The quality of the acquired image is fundamental: any distortion, blur, or poor lighting can cause problems throughout the rest of the OCR pipeline.

To achieve this, we need to ensure the image is in focus and undistorted. After that to save the image in a lossless or high-quality format (e.g., PNG, TIFF, high-quality JPEG) to preserve fine details.

2.2.2. Preprocessing

Once the image is acquired, preprocessing is done to enhance its quality and prepare it for analysis. The goal is to make the text clearer and more uniform for further steps. Good preprocessing produces a cleaner, more consistent image and directly improves the accuracy of segmentation and recognition.

Best preprocessing practices are:

Converting color to grayscale: reduces data complexity and focuses on text content.

Binarization (thresholding): converts the grayscale image to black-and-white, making text stand out from the background.

Noise removal: eliminates random dots or specks that could be mistaken for text.

Skew correction: straightens slanted text lines for proper alignment.

Morphological operations: Erosion/Dilation to adjust line thickness or separate joined elements. Opening/Closing to clean up isolated noise or fills small gaps.

Compression: may be applied for storage, but should avoid quality loss.

2.2.3. Character Segmentation

Segmentation splits the processed image into individual characters (or sometimes words). This stage is critical, as joining or splitting characters incorrectly can cause major recognition errors.

Simple cases: connected component analysis, projection profiles.

Complex cases: advanced algorithms for overlapping, touching, or broken characters, especially in handwritten or degraded texts.

Proper segmentation ensures each character is analyzed independently. Errors at this step often result in misclassification, since the recognition engine receives the wrong input.

2.2.4. Feature Extraction

Each character image is described using features that help distinguish it from other characters. The effectiveness of this stage depends on selecting features that highlight differences between classes and minimize within-class variation.

Type of features:

Geometric: number of loops, endpoints, aspect ratio, stroke directions.

Statistical: pixel distributions, statistical moments, zoning (dividing the character into regions and measuring pixel densities).

Transform-based: features from projections, Fourier or Wavelet transforms.

Dimensionality reduction: techniques such as Principal Component Analysis (PCA) to keep only the most important features.

Poorly chosen features make it hard to distinguish similar characters, reducing classification accuracy.

2.2.5. Classification

The extracted features are now used to identify (classify) each character. This is the core step where the actual recognition happens. Classification approaches are:

Structural classifiers: use explicit rules based on shape and structure.

Statistical classifiers: data-driven methods such as Bayesian classifiers, decision trees, neural networks, nearest neighbor algorithms.

Syntactic classifiers: treat a character as a sequence of basic shapes, similar to grammar parsing.

The classifier's accuracy depends on the quality of the previous steps. Even a strong classifier can't compensate for poor segmentation or weak features.

2.2.6. Post-Processing

After initial classification, post-processing is used to correct errors and refine the output. This is especially important for real-world documents and complex languages.

Methods include:

Spell checking and dictionary lookup: corrects misrecognized words.

Language models (n-grams, Markov chains): predict the most probable sequence of characters or words.

Geometric and layout analysis: uses document structure (like columns or tables) to resolve ambiguities.

Combining classifiers: multiple classifiers can work together (cascaded, parallel, or hierarchical), and their outputs are combined for better reliability.

Post-processing helps improve the practical usefulness and reliability of OCR output, especially for noisy or ambiguous input.

These steps help improve the practical usefulness and reliability of OCR output, especially for noisy or ambiguous input. [4]

2.3. Document Understanding Models

2.3.1. Tesseract OCR

Tesseract is a classical open-source OCR engine initially developed by Hewlett-Packard and later open-sourced by Google. Its architecture is based on a step-by-step pipeline: the engine first analyzes a binary image to identify and group connected components (contours) into 'blobs,' which are then organized into text lines and words. Tesseract supports both fixed-pitch and proportional fonts and employs adaptive classification, where successful word recognitions are used to refine the model further as it processes the page.

One of the distinguishing features of Tesseract is its ability to recognize text on skewed or curved baselines without requiring explicit deskewing, which helps preserve image quality. It can also handle both black-on-white and white-on-black text. Tesseract applies a combination of segmentation (splitting joined letters and associating broken characters) and static/adaptive classification to improve recognition on noisy or degraded images. The linguistic module is minimal, focusing primarily on basic dictionary and frequency checks.

Tesseract has demonstrated high accuracy for printed text in standard benchmark tests (such as the UNLV Annual Test of OCR Accuracy). However, it does not provide built-in page layout analysis—users must supply segmented text regions. [5]

Advantages:

- Open-source, accurate on clean printed text
- Handles skewed and slightly degraded images
- Fast and easy to integrate

Limitations:

- No built-in layout or table analysis
- Poor at checkboxes, handwriting, or complex receipts
- Often needs extra postprocessing for structured data

Conclusion: Tesseract is suitable for extracting text from high-quality printed receipts, but for real-world photos, varied templates, or receipts with marks, it should be combined with preprocessing and/or deep learning models for best results.

2.3.2. EasyOCR

EasyOCR is an open-source OCR library written in Python and implemented using PyTorch. It supports more than 80 languages, including English and Spanish. The EasyOCR pipeline consists of two main stages: detection and recognition. The detection stage uses the CRAFT model to locate text regions and bounding boxes in the input image.

The recognition stage is based on a CRNN model and consists of three components:

1) feature extraction using CNNs (e.g., ResNet, VGG);
2) sequence labeling with an RNN such as LSTM to interpret sequential context
3) decoding the sequence of probabilities into text using the Connectionist Temporal Classification (CTC) algorithm.

This design allows EasyOCR to handle a variety of document types and natural scene images, working robustly even with distortions or challenging layouts.[6]

Advantages:

- Open-source, easy to use
- Good support for many languages
- Robust to different text orientations and some distortions

Limitations:

- Not specialized for table/checkbox extraction or highly structured layouts
- Can lose accuracy on distorted or multi-oriented text

Conclusion: EasyOCR is suitable for extracting text from real-world images and receipts, but for complex other models may provide better results.

2.3.3. PaddleOCR

PaddleOCR is an open-source OCR toolkit based on the PaddlePaddle deep learning framework. The latest versions (PP-OCRv2, PP-OCRv3) feature a three-stage pipeline:

1) text detection using Differentiable Binarization, a segmentation network trained with collaborative mutual learning.

2) rectification of detected text boxes to standardize orientation, including reversal detection.

3) text recognition using SVTR-LCNet, which fuses Transformer-based and lightweight CNN-based architectures for efficient and accurate sequence recognition.

PaddleOCR is designed for versatility and high performance in a range of environments, including standard documents and challenging natural scenes. It offers competitive results even with distorted, curved, or multi-oriented text.[6]

Advantages:

- High accuracy, especially for distorted, curved, or multi-oriented text
- Advanced detection and recognition pipeline
- Scalable and efficient (multiple lightweight and full models)

Limitations:

- More complex to deploy and configure compared to some other libraries
- Requires more resources for advanced model

Conclusion: PaddleOCR is a powerful OCR library for extracting text from both standard and challenging real-world images. Its robust pipeline makes it especially suitable for complex layouts and distorted scenes, making it a top choice for automated document processing tasks. However, for tasks requiring precise layout analysis, checkbox detection, or high-accuracy structured extraction, it may need to be combined with additional tools or models.

2.3.4. Donut

Donut (Document Understanding Transformer) is an end-to-end, OCR-free model for visual document understanding. Unlike conventional pipelines that rely on separate OCR engines for text extraction, Donut uses a transformer-based architecture to directly map document images to structured outputs, such as JSON. The model consists of a visual encoder (based on Swin Transformer) and a text decoder (based on BART), trained together to extract and structure information from documents in a single step.

Donut can be pre-trained on large-scale synthetic and real document datasets, and fine-tuned for specific downstream tasks, such as receipt parsing, invoice extraction, and document classification. Its design makes it robust to complex layouts, multiple languages, and noisy real-world images. Donut is especially effective when fine-tuned on examples matching the specific document formats and target outputs needed in your workflow (including checkboxes or manual marks, if they are present in the annotation). [7]

Advantages:

- End-to-end extraction of structured information (e.g., JSON) directly from images
- No need for a separate OCR engine or post-processing
- High accuracy and efficiency on diverse document types, including receipts, invoices, and forms
- Robust to complex layouts, multiple languages, and visual elements (when present in the annotation)

Limitations:

- Requires substantial computational resources for training and inference
- Out-of-the-box accuracy may be low without fine-tuning on relevant data
- May miss tiny or very low-resolution objects if input images are not sufficiently detailed

Conclusion: Donut offers a modern, flexible approach for automated document processing, enabling direct extraction of structured data including text, checkboxes, and manual marks from receipts and forms. For best results, it should be fine-tuned on a dataset representative of your target documents and annotation schema.

2.3.5. TrOCR

TrOCR (Transformer-based OCR) is an end-to-end text recognition model that leverages pre-trained Vision and Language Transformers. Unlike traditional OCR approaches based on CNN and RNN backbones, TrOCR uses a pure Transformer architecture both for visual feature extraction (encoder) and language modeling (decoder). The encoder processes the input image as a sequence of patches (like in Vision Transformer, ViT), while the decoder generates the recognized text as a sequence of wordpiece tokens, eliminating the need for post-processing or external language models.

TrOCR is pre-trained on large-scale synthetic and real datasets, including printed, handwritten, and scene text, and can be fine-tuned for various document types. It achieves state-of-the-art results on multiple benchmarks (printed receipts, handwritten forms, and scene text) with high robustness to noise and distortion.[8]

Example output (flat text):

Coca-Cola 1.25 / Bread 2.50 / Butter 3.20 / TOTAL 6.95

Advantages:

- End-to-end text recognition without the need for CNNs or RNNs
- Utilizes pre-trained Transformer models for both image and text, enhancing accuracy
- Achieves high performance on printed, handwritten, and scene text tasks
- Easily extensible for multilingual OCR

Limitations:

- Does not output structured data - produces only flat text sequences
- Requires cropping text lines or regions for optimal performance
- Needs substantial computational resources for training and large model variants

Conclusion: TrOCR provides a modern, robust solution for OCR tasks in diverse conditions, excelling at both printed and handwritten text. It is best suited for scenarios where precise text transcription is needed, but structured extraction (e.g., field parsing) may require additional post-processing or integration with other models.

2.3.6. Summary

Recent advances in OCR and document understanding have dramatically improved the extraction of information from receipts and other administrative documents. Traditional engines such as Tesseract remain reliable for clean, printed text but struggle with real-world photos, complex layouts, or documents containing handwritten or marked elements. Deep learning-based tools like EasyOCR and PaddleOCR offer improved robustness for natural scenes, distortions, and multilingual content, with PaddleOCR providing superior accuracy and versatility for challenging inputs.

End-to-end transformer models (Donut and TrOCR) represent the next generation of document understanding. Donut enables direct extraction of structured data from images. TrOCR excels at robust text transcription in a wide range of scenarios.

Overall, each model has its unique strengths:

- **Tesseract** is best for standard, high-quality printed receipts;
- **EasyOCR** and **PaddleOCR** address more variable, real-world images and languages;
- **Donut** provides direct, structured information extraction while **TrOCR** delivers high-accuracy text recognition across diverse document types

2.4. Checkmarks recognition

Checkmark detection is a specific sub-task within object detection, a broader area of computer vision that involves locating and classifying various visual elements in images or documents. In the case of checkmark detection, the goal is to automatically identify checkboxes or hand-drawn check marks on documents, such as receipts or forms, to determine whether a particular item is selected. This capability is important for interpreting user intent, for example, when distinguishing reimbursable expenses from personal purchases.

Unlike character recognition (OCR), which extracts alphanumeric information, checkmark detection focuses on identifying selection markers that represent a binary decision - selected or not selected. Because these markers are not standard text, checkmark detection relies on object detection algorithms rather than traditional OCR techniques, enabling accurate interpretation of non-textual selection marks.

2.4.1. Object Detection

Object detection is a core computer vision task that involves not only classifying objects within an image, but also precisely localizing them by predicting bounding boxes. Unlike simple image classification, object detection must solve both “what” and “where” for potentially multiple objects of different types in the same image.

Key Concepts

- **Bounding Box:** The rectangular region that encloses an object, usually defined by (x, y, width, height) or (xmin, ymin, xmax, ymax).
- **Class Label:** The predicted category (e.g., checkmark, background) assigned to each detected object.
- **Confidence Score:** The model’s estimated probability that a bounding box contains an object of a given class.
- **IoU (Intersection over Union):** A metric for evaluating the overlap between the predicted and ground truth bounding boxes, widely used in object detection benchmarks.

Main Approaches

- **Two-stage Detectors:** First generate region proposals, then classify and refine bounding boxes (e.g., R-CNN, Fast R-CNN, Faster R-CNN).
- **One-stage Detectors:** Directly predict bounding boxes and class labels for all locations in a single forward pass (e.g., YOLO, SSD).
- **Anchor Boxes:** Predefined bounding boxes of various sizes and ratios, used to help networks detect objects of different scales and aspect ratios.

Evaluation Metrics

- **mAP (mean Average Precision):** Standard benchmark for detection accuracy, averaging precision across all object classes and IoU thresholds.
- **FPS (Frames per Second):** Standard for measuring inference speed of detection models.

Applications

Object detection methods are used in a variety of fields, including automated document analysis, face and pedestrian detection, traffic surveillance, and medical image analysis. In this work, object detection is applied to the recognition and localization of checkmarks in receipt images, enabling structured expense tracking and digital record creation. [9]

2.4.2. Brief History of Object Detection Methods

Object detection has evolved from early approaches based on handcrafted features (such as SIFT, HOG, Haar-like) and shallow models (like SVM and DPM), to deep learning-based methods. Early deep detectors (R-CNN, 2014) used CNNs to classify region proposals generated by external algorithms. Fast R-CNN and SPP-net introduced architectural improvements to speed up detection by sharing convolutional computations and enabling multi-scale analysis. Faster R-CNN integrated a Region Proposal Network (RPN) for generating proposals, making the pipeline end-to-end trainable. One-stage detectors, such as YOLO and SSD, further improved detection speed by removing the proposal step and directly predicting object locations and classes from the full image in a single pass. Recent advances include architectures like Mask R-CNN for instance segmentation and FPN for better multi-scale object detection.[9]

Table 3. Comparison of Key Deep Learning Object Detection Methods

Method	**Architecture Summary**	**Speed/Accuracy**	**Suitable?**
R-CNN	2-stage: region proposals + CNN +SVM classification	~0.03 fps mAP = 66%	**No:** too slow, has complex pipeline.
Fast R-CNN	2-stage: shared conv layers, RoI pooling, SVM/SOFTmax	~0.6 fps mAP = 70%	**No**: still slow
Faster R-CNN	2-stage: RPN for proposals, end-to-end trainable	2-9 fps mAP = 70-78%	**Yes**
YOLO (v1-v3)	1-stage: direct regression, grid-based, real-time	40-60 fps mAP = 57-80%	**No:** not supported anymore
YOLO (v8)	1-stage: anchor-free, decoupled head	50-120 fps mAP = 85-90%	**Yes**
SSD	1-stage: multi-scale feature maps, anchors	19–46 fps mAP = 74–82%	**Yes**
Mask R-CNN	2-stage: Faster R-CNN + instance segmentation	5–6 FPS mAP = 78–82%	**No:** only for objects' masks

YOLOv8 and SSD appear optimal for checkmark detection tasks due to their high speed, accuracy, and flexibility. Faster R-CNN is also relevant, especially for precise detection of smaller marks. Therefore, I will focus further analysis and experimentation on YOLO, SSD, and Faster R-CNN.

2.4.3. YOLO

One of the most widely used and efficient methods for object detection—including the specific task of checkmark recognition in receipts—is the YOLO (You Only Look Once) algorithm. Originally proposed by Joseph Redmon et al. (2016), YOLO has become a fundamental tool in various domains requiring real-time object detection and robust generalization.

YOLO reframes object detection as a single regression problem: the input image is divided into an S×S grid, where each grid cell is responsible for detecting objects whose centers fall within it. For each cell, the neural network predicts parameters of B bounding boxes (coordinates, width, height, and confidence) as well as C class probabilities. This is accomplished in a single forward pass of a convolutional neural network, resulting in extremely high inference speed (up to 45–155 frames per second on GPU), making YOLO highly suitable for real-time applications such as automated receipt processing.

For the specific task of checkmark recognition in receipts, YOLO can be used both for binary classification (present/absent) and for more complex labeling (e.g., different mark types). The network is trained on a synthetic dataset of checkmarks, where each annotated sample contains bounding box coordinates and the corresponding class. The model outputs structured bounding boxes with class probabilities, which can be directly post-processed into digital records or tables for subsequent analysis of receipt data.[10]

Advantages:

- Real-time detection (up to 45–155 images/sec on GPU)
- No manual cropping or handcrafted features
- Easily adapts to various checkmark styles
- Outputs structured results (boxes, scores)
- Reduces false positives via full-image context

Limitations:

- Accuracy drops for very small or clustered checkmarks
- Sensitive to shapes/noise not seen in training
- Needs GPU for high speed, CPU is slow
- Relies on diverse synthetic training data

Conclusion:

YOLO provides a fast and reliable solution for automated checkmark recognition in receipts, particularly well-suited for high-throughput or real-time processing scenarios. While it excels at extracting structured data from typical receipt layouts, extra care should be taken when dealing with densely clustered, extremely small, or visually unusual checkmarks. Overall, YOLO is a robust foundation for building practical pipelines that transform raw receipt scans into digital expense records with minimal delay.

2.4.4. Faster R-CNN

Faster R-CNN is a two-stage object detection framework that has become a standard for accuracy-oriented detection tasks. It integrates a Region Proposal Network (RPN) for generating candidate object regions and a Fast R-CNN detector for refining and classifying those regions within a single, unified neural network. The convolutional layers are shared between the proposal and detection stages, making the approach both efficient and end-to-end trainable.

For checkmark recognition in receipts, Faster R-CNN can be trained to detect and classify small marks with high precision, even when checkmarks are close together or partially degraded. The method is fully open-source and can be adapted to any custom dataset, including synthetic checkmarks or real scanned receipts.[11]

Advantages:

- High detection accuracy, including for small or densely packed checkmarks
- Robust to noise, complex backgrounds, and overlapping objects
- Flexible: supports custom classes and anchor settings for checkmark detection
- Open-source, free to use and modify for research and commercial projects
- Works well with limited or imbalanced data thanks to region proposal mechanism

Limitations:

- Slow inference speed (typically 2–7 images/sec on GPU)
- Requires more computational resources for both training and inference
- More complex pipeline to set up and fine-tune compared to single-stage detectors
- Not ideal for scenarios with strict real-time speed constraints

Conclusion:

Faster R-CNN is an excellent choice for scenarios where detection accuracy and robustness are critical, such as receipts with many overlapping or faint checkmarks. It is particularly suitable when processing speed is not the primary constraint, and when maximum precision is required. Thanks to open-source implementations and customizable architecture, Faster R-CNN is accessible and practical for tailored receipt analysis tasks.

2.4.5. SSD

Single Shot MultiBox Detector (SSD) is a one-stage object detection algorithm designed to balance high speed and accuracy. Introduced by Liu et al. (2016), SSD predicts bounding boxes and class probabilities directly from multiple feature maps of different scales in a single network pass, without any region proposal step. This design allows SSD to efficiently handle objects of varying sizes in real-time applications.

Key features of SSD:

- Uses a set of default (anchor) boxes of various aspect ratios and scales at each feature map location.
- Combines predictions from multiple feature maps for better multi-scale detection.
- Enables end-to-end training and inference with a relatively simple pipeline.

For checkmark recognition in receipts, SSD can be easily adapted and trained on a custom synthetic dataset. It is able to detect and localize checkmarks with high speed and competitive accuracy, making it a practical solution for automated receipt processing and expense extraction. [12]

Advantages:

- Real-time inference: up to 46–59 receipts/sec on GPU (SSD300), higher for smaller images
- Good detection accuracy (up to 74–77% mAP on PASCAL VOC)
- Handles objects of different sizes via multi-scale predictions
- Simple end-to-end architecture, easy to integrate and train
- Open-source and widely supported

Limitations:

- Performance drops for very small or densely packed checkmarks
- Sensitive to the quality of anchor box design and data augmentation
- Accuracy slightly below two-stage detectors on challenging cases

Conclusion:

SSD offers an excellent balance between speed and accuracy for automated checkmark detection in receipts. It’s simple one-stage design and support for real-time processing make it a strong candidate for large-scale or high-throughput systems.

2.5. Existing solutions

In addition to academic and open-source approaches, a variety of commercial products and startups offer automated receipt recognition as part of expense management or business workflow automation. Some of the most prominent solutions include:

Expensify

A widely used SaaS platform for expense management, Expensify allows users to photograph receipts with a mobile app. The system extracts key information such as merchant, date, and amount using proprietary OCR and post-processing algorithms. While it excels at standard receipt layouts and automates much of the reporting process, Expensify does not provide access to the underlying extraction pipeline and offers limited support for custom document formats or user-added marks such as checkboxes.[13]

ABBYY FlexiCapture

An enterprise-level document recognition solution, ABBYY FlexiCapture combines advanced OCR, template-based extraction, and machine learning. It supports a wide range of documents including receipts and invoices, allowing for some customization of field extraction. However, integration and configuration can be complex, and the system works best with pre-defined, standard templates. Detection of manual marks, checkboxes, or ad hoc annotations is not natively supported.[14]

Veryfi

Veryfi provides a developer-oriented API for rapid receipt and invoice data extraction. The solution relies on proprietary AI models and cloud infrastructure, returning structured JSON with extracted fields. Veryfi is known for high accuracy and processing speed with standard receipts, but does not expose internals or allow for fine-tuning on custom marks or checkboxes.[15]

Dext (formerly Receipt Bank), Shoeboxed, WellyBox, Gorilla Expense

These SaaS tools and mobile apps target small business and individual users. They typically extract merchant, date, and amount, sometimes line items, and provide interfaces for manual correction. However, they are primarily designed for mainstream receipt formats and do not support custom annotations or automated selection of marked items.. [16] [17] [18] [19]

Table 4. Existing solutions comparison

Solution	Structured Data Export	Custom Layouts	Checkmark recognition	Pricing Policy
Expensify	Yes	No	No	Subscription, from $5/user/month
ABBYY FlexiCapture	Yes	Partial (templates)	No	Commercial, custom pricing
Veryfi	Yes (API)	No	No	Pay-per-use/API credits
Dext, Shoeboxed, etc.	Yes	No	No	Subscription, varies
Open-source pipeline (e.g. Tesseract + YOLO)	Yes (custom)	Yes (customizable)	Yes (with training)	Free (open-source), costs only for infra/training

2.6. Gap in existing approaches

Despite their ease of use, most commercial platforms are “black-box” solutions. They rarely support automated recognition of user-added checkmarks or arbitrary item selection. Customization is typically limited to template configuration or manual correction, making these platforms unsuitable for scenarios that require robust checkmark/checkbox extraction or downstream structuring of data according to user-defined marks.

Moreover, some of platforms, like Shoeboxed “go a step further” by human-verifying all extracted data which does not match the word “automation” at all.

Summary:

Commercial receipt recognition solutions are convenient for standard expense management but are generally unsuitable for advanced scenarios that require reliable detection of checkmarks, custom fields, or direct integration with research workflows. Open-source and custom pipelines, while requiring more development effort, offer the necessary flexibility and control for these tasks.

3. Problem statement

3.1. Description of the Use Cases

Private case: going to a restaurant with friends. The user pays a common bill for the table. He manually marks his items on the paper check. After that, the user takes a photo of the check and sends the image along with the final amount (to check the data accuracy).

Business case: accounting for a yacht captain. The captain of a long-distance yacht regularly makes purchases, paying for expenses personally. The captain's personal expenses that are not subject to compensation are marked with checkmarks. The captain sends the accounting department photos of several checks with marked items and each check’s total sum.

As a result, the system should automatically recognize all items, exclude the lines marked with checkboxes from the final table and also check that the amount calculated for all items matches the amount entered by the user.

3.2. Requirements

Input data: photograph of paper receipt with checkmarks and the amount entered by the user for verification. Photos should be done on the black background.

Receipt language: German.

Result: a structured table of all expenses minus the items marked by the user plus a final sum.

Validation: mandatory match of the total amount calculated by the system with the amount declared by the user.

Success criterion: maximum accuracy of information extraction. Priority is given to recognition quality, processing time is secondary

Limitations: different types of receipts are allowed; photo quality requirements are standard. The receipt must be legible and not too deformed.

4. Methodology

4.1. Dataset for Checkmark Recognition

4.1.1. Handmade Dataset

The initial dataset was created by manually collecting and annotating images of checkmarks. Each image was labeled in YOLO format, specifying the coordinates of each checkmark using bounding boxes. Manual annotation was performed using makesense.ai, a web-based tool that supports direct export to YOLO format. This dataset included a set of real-world checkmarks with variations in size and style.

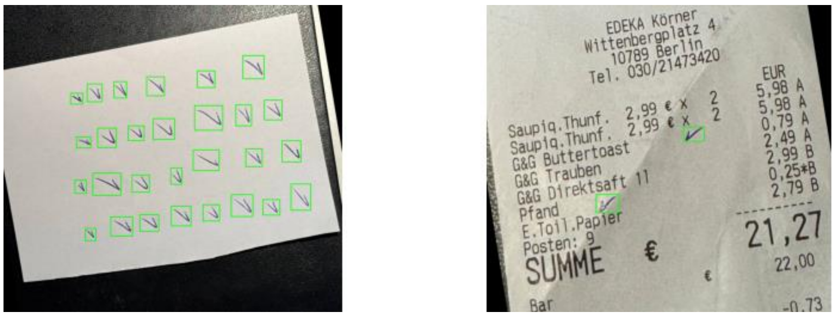


Figure 2. Examples of handmade checkmarks with bounding box annotation.

As for annotation format, labels were saved in YOLO format as plain text files. Each line corresponds to a bounding box (normalized values):

Table 5. Examples of a YOLO annotation line

class_id	**x_center**	**y_center**	**width**	**height**
0	*0.615328*	*0.441522*	*0.054902*	*0.025237*
0	*0.335946*	*0.561068*	*0.059330*	*0.033207*

Data augmentation was applied to the handmade images using the Albumentations library. Key augmentations included horizontal flipping and small-angle rotation. The code fragment below shows the augmentation pipeline:

```
transform = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.Rotate(limit=15, border_mode=0, p=1.0)
],                                  bbox_params=A.BboxParams(format='yolo',
label_fields=['category_ids']))
```

Listing 1. Augmentation parameters used for the handmade checkmark dataset.

4.1.2. Synthetic Dataset

To further increase the size and variability of the dataset, a synthetic checkmark generator was implemented in Python. The generator creates images by drawing checkmarks with randomized parameters: starting position, length and angle of each line and thickness. The size of each generated image was fixed at 224×224 pixels.

```python
def draw_checkmark(img):
    x1, y1 = random.randint(30, 80), random.randint(120, 170)
    x2, y2 = x1 + random.randint(8, 25), y1 + random.randint(20, 50)
    x3, y3 = x2 + random.randint(12, 30), y2 - random.randint(30, 60)
    thickness = random.randint(3, 6)
    color = random.randint(0, 30)
    cv2.line(img, (x1, y1), (x2, y2), color, thickness)
    cv2.line(img, (x2, y2), (x3, y3), color, thickness)
    xmin, xmax = min(x1, x2, x3), max(x1, x2, x3)
    ymin, ymax = min(y1, y2, y3), max(y1, y2, y3)
    x_center = ((xmin + xmax) / 2) / img_size
    y_center = ((ymin + ymax) / 2) / img_size
    w = (xmax - xmin) / img_size
    h = (ymax - ymin) / img_size
    return x_center, y_center, w, h
def gen_images(count, subset):
    for i in tqdm(range(count), desc=f'Generation {subset}'):
        img = np.ones((img_size, img_size), dtype=np.uint8) * 255
        bbox = draw_checkmark(img)
        img_name = f'checkmark_{i:04d}.jpg'
        label_name = img_name.replace('.jpg', '.txt')
        cv2.imwrite(f'{base_dir}/images/{subset}/{img_name}', img)
        with open(f'{base_dir}/labels/{subset}/{label_name}', 'w') as f:
            f.write(f'0 {bbox[0]:.4f} {bbox[1]:.4f} {bbox[2]:.4f}
{bbox[3]:.4f}')
```

Listing 2. Functions for image generation and synthetic checkmarks creation

A total of 160 synthetic images were generated for training and 40 for validation. Each checkmark was automatically annotated and saved in YOLO format.

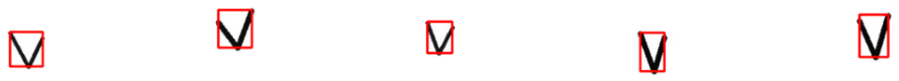


Figure 3. Examples of synthetically generated checkmarks and bounding boxes

4.1.3. Final Dataset Assembly

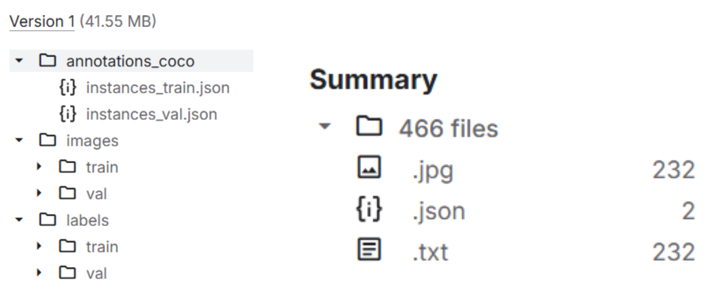
The entire dataset was also converted to COCO format for compatibility with different detection frameworks. The COCO (Common Objects in Context) format organizes dataset information in a single JSON file with the following key sections:

Images: info about each image (id, filename, size)
Annotations: bounding box coordinates and class for each object
Categories: list of object classes (id and name)

```
{  "images": [
    {"id": 1, "file_name": "checkmark_0000.jpg", "width": 224, "height": 224},
    {"id": 2, "file_name": "handmade_01.jpg", "width": 1920, "height": 2560}
  ],
  "annotations": [
    {"id": 1, "image_id": 1, "category_id": 1, "bbox": [35.0, 142.9, 42.0, 32.0], "area": 1344.4, "iscrowd": 0},
    {"id": 2, "image_id": 2, "category_id": 1, "bbox": [1126.8, 747.4, 73.2, 63.2], "area": 4625.5, "iscrowd": 0}
  ],
  "categories": [
    {"id": 1, "name": "checkmark"}
  ]}
```

Listing 3. Part of COCO annotation .json file

After combining all sources and augmentations, the final dataset consisted of about **400** training checkmarks and **100** validation checkmarks. Each image is accompanied by a YOLO-format annotation file and COCO annotations.



Version 1 (41.55 MB)

- annotations_coco
  - instances_train.json
  - instances_val.json
- images
  - train
  - val
- labels
  - train
  - val

Summary

466 files

.jpg	232
.json	2
.txt	232

Figure 4. Hierarchy and structure of final checkmarks Dataset

The resulting dataset provides a balanced mix of realistic and artificially generated checkmarks. The link to complete dataset, including all images and annotation files, is available at Appendix A. Reproducibility and Public Access

4.2. Checkmark Recognition Models Comparison

4.2.1. YOLO

YOLOv8n is a fast and lightweight one-stage detector, suitable for resource-constrained and speed-critical tasks.

```
model = YOLO('yolov8n.pt')
result = model.train(
    data=yolo_dataset_yaml,
    epochs=20,
    imgsz=640,
    project='/kaggle/working/yolov8_checkmark_run',
    name='exp',
    workers=2)
```

Listing 4. YOLOv8n initialization and training

YOLOv8n was trained for 20 epochs on 640x640 images, using the official Ultralytics API. The training automatically applies data augmentations and uses the SGD optimizer with adaptive learning rate. This setup is optimal for fast prototyping and high performance even on relatively small datasets.

4.2.2. Faster R-CNN

```
frcnn_model =
torchvision.models.detection.fasterrcnn_resnet50_fpn(num_classes=2,
pretrained=False).to(device)
frcnn_model.train()
frcnn_optimizer = torch.optim.Adam(frcnn_model.parameters(), lr=1e-4)
for epoch in range(5):
    for images, targets in train_loader_coco:
        images = [img.to(device) for img in images]
        targets = [{k: v.to(device) for k, v in t.items()} for t in
targets]
        loss_dict = frcnn_model(images, targets)
        loss = sum(loss for loss in loss_dict.values())
        frcnn_optimizer.zero_grad()
        loss.backward()
        frcnn_optimizer.step()
    print(f"Epoch {epoch+1}: Faster R-CNN Loss: {loss.item():.4f}")
```

Listing 5. Faster R-CNN initialization and training

Faster R-CNN ResNet50-FPN is a two-stage detector with high accuracy, especially good for small objects, but slower and more resource-demanding.

Faster R-CNN was trained for 5 epochs with batch size 4 and Adam optimizer, learning rate 1e-4. Number of epochs was reduced due to higher computational demands.

4.2.3. SSD

SSD300-VGG16 is a compromise between speed and quality, commonly used when speed is important and the objects are of medium or large size.

```
ssd_model = torchvision.models.detection.ssd300_vgg16(num_classes=2,
pretrained=False).to(device)

ssd_model.train()

ssd_optimizer = torch.optim.Adam(ssd_model.parameters(), lr=1e-4)

for epoch in range(10):

    for images, targets in train_loader_coco:

        images = [img.to(device) for img in images]

        targets = [{k: v.to(device) for k, v in t.items()} for t in
targets]

        loss_dict = ssd_model(images, targets)

        loss = sum(loss for loss in loss_dict.values())

        ssd_optimizer.zero_grad()

        loss.backward()

        ssd_optimizer.step()

    print(f"Epoch {epoch+1}: SSD Loss: {loss.item():.4f}")
```

Listing 6. SSD300-VGG16 initialization and training

SSD300-VGG16 was trained for 10 epochs with a batch size of 4 and Adam optimizer, learning rate 1e-4. No pretraining was used to ensure fair comparison. The batch size is kept modest for stability on limited GPU memory.

4.2.4. Metric Comparison and Visualization

Table 6. Checkmark recognition models comparison

Model	**mAP@0.5**	**Precision**	**Recall**
YOLOv8n	0.995	0.999	1.000
SSD300-VGG16	0.693	0.693	0.650
Faster R-CNN	0.296	0.296	1.000

All models were trained and validated on the same dataset. For YOLO the built-in .val() method was used. For SSD/Faster R-CNN a custom calc_map function.

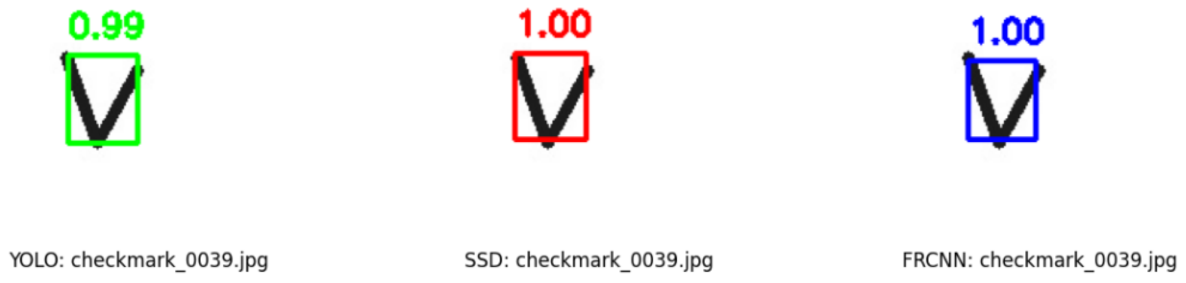


Figure 5. Generated checkmark detection comparison

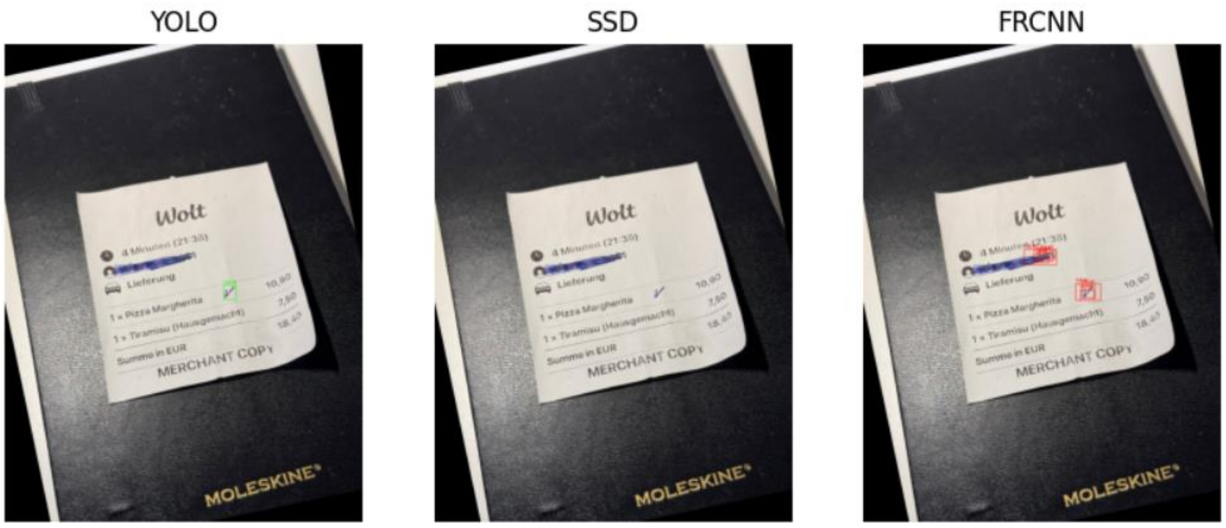


Figure 6. Handmade checkmark detection comparison

4.2.5. Conclusion

YOLOv8n demonstrated the best results in terms of accuracy (mAP@0.5) and overall checkmark localization quality. Moreover, YOLOv8n has much better results in non-generated checkmark recognition. That’s why for practical checkmark detection tasks on receipts, **YOLOv8n** was recommended as the primary model.

Link to the source code can be found at Appendix: Code Access and Source Examples

4.3. Image Acquisition and Preprocessing

The image acquisition and preprocessing logic was separated into a standalone Python module. This approach allows the main system to simply request a ready-to-use receipt image without handling low-level image processing routines. Such separation simplifies future updates of preprocessing methods.

4.3.1. Image Load

The pipeline begins by obtaining the image of the receipt. The script first downloads the image and optionally resizes it so that its maximum side does not exceed 2048 pixels. This ensures consistent input dimensions and manageable resource usage.

```python
def download_image(url=receipt_url, out_path=receipt_path, resize=True,
max_width=2048, max_height=2048):
    gdown.download(url, out_path, quiet=False)
    if resize:
        img = cv2.imread(out_path)
        img = smart_resize(img, max_width=max_width,
max_height=max_height)
        cv2.imwrite(out_path, img)
```

Listing 7. Download function

4.3.2. Image Preprocessing

Further steps are:

Grayscale Conversion: the image is converted to grayscale to simplify further operations and reduce noise from color artifacts.

Binarization: Otsu's thresholding is applied to obtain a binary image, separating foreground (text, marks) from the background.

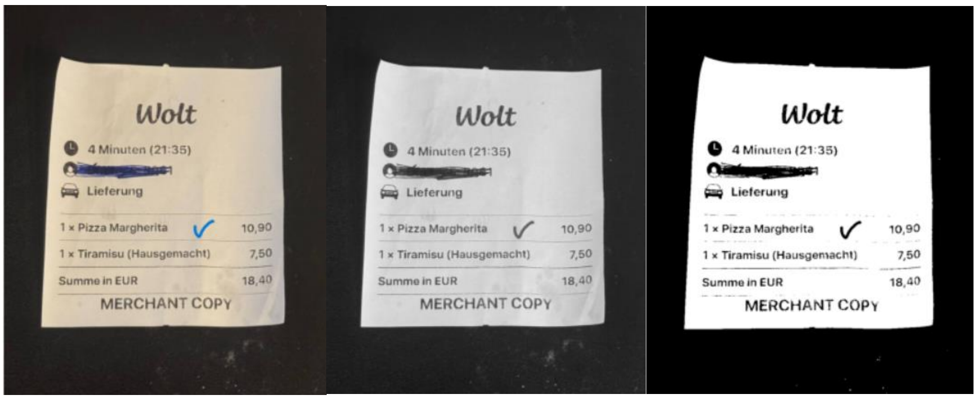


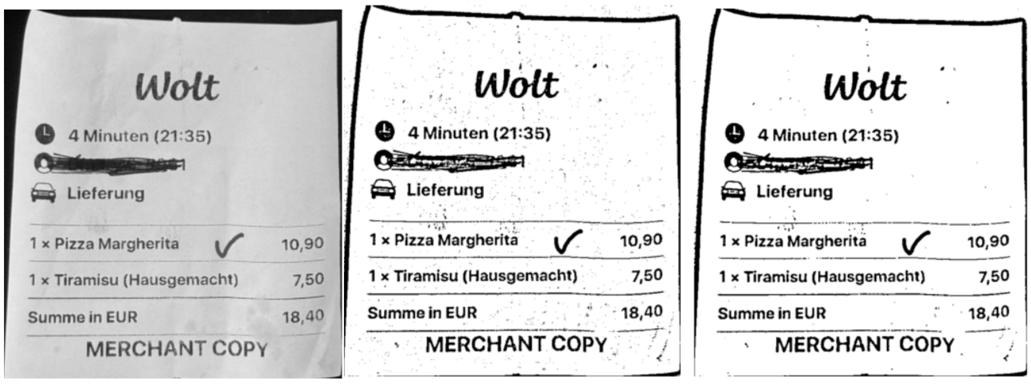
Figure 7. Original -> Grayscale -> Otsu Binarized

Deskew (not implemented) - autocorrection of the rotation of a receipt. Stable deskew was not successfully implemented, and remains a potential improvement.

Cropping - the algorithm detects the largest external contour (assumed to be the receipt) and crops the image to this region, removing any background.

Adaptive Thresholding - adaptive binarization further enhances the contrast of text and checkboxes, making them more distinguishable for OCR and detection.

Noise Reduction - a median blur filter is applied to remove small artifacts.



Wolt

4 Minuten (21:35)

[redacted]

Lieferung

1 × Pizza Margherita ✓	10,90
1 × Tiramisu (Hausgemacht)	7,50
Summe in EUR	18,40

MERCHANT COPY

Figure 8. Cropped -> Thresholded -> Denoised

4.3.3. Conclusion

It was observed that OCR performance was often higher on original, unprocessed receipt images compared to preprocessed images. That's why, I decided to implement two simple functions to obtain either a clean (raw) or preprocessed version of the receipt image.

```
def get_preprocessed_receipt(url, out_path=receipt_path):
    download_image(url, out_path)
    return preprocess_receipt(out_path)


def get_raw_receipt(url, out_path=receipt_path):
    download_image(url, out_path)
    return cv2.imread(out_path)
```

Listing 8. Get receipt functions

Link to the source code can be found at Appendix: Code Access and Source Examples

4.4. Text recognition models

4.4.1. Donut (not suitable)

Donut is an end-to-end transformer model for extracting structured data from documents. It works without a classical OCR engine and produces structured output.

```python
def process_receipt_donut(image_path):
    image = Image.open(image_path).convert("RGB")
    task_prompt = "<s_cord-v2>"
    pixel_values = processor(image, 
return_tensors="pt").pixel_values.to(device)
    outputs = model.generate(
        pixel_values,
        decoder_input_ids=processor.tokenizer(task_prompt, 
add_special_tokens=False, return_tensors="pt").input_ids.to(device),
        max_length=model.decoder.config.max_position_embeddings,
        early_stopping=True,
        pad_token_id=processor.tokenizer.pad_token_id,
        eos_token_id=processor.tokenizer.eos_token_id,
        use_cache=True,
        num_beams=1,
    )
    sequence = processor.batch_decode(outputs, 
skip_special_tokens=True)[0]
    result = processor.token2json(sequence)
    if isinstance(result, str):
        result = json.loads(result)
    return result
processor = DonutProcessor.from_pretrained("naver-clova-ix/donut-base-
finetuned-cord-v2")
model = VisionEncoderDecoderModel.from_pretrained("naver-clova-ix/donut-
base-finetuned-cord-v2")
result = process_receipt_donut(receipt_path)
donut_to_dataframe(result)
```

Listing 9. Donut initialization + image parsing function

This model was considered the most promising solution at the preparation stage. Unfortunately, in practice, it failed to correctly recognize even simple receipts. It added “minutes” and merchant copy as items, missed Tiramisu and got pizza’s wrong price. The main goal of this project is to target a wide variety of receipt types. Donut requires additional fine-tuning on specific receipt examples, which is not suitable for this case.

Table 7. Donut recognition results

Item	Count	Price
Minuten (21:35)	1 x	
Pizza Margherita	1 x	18,40
MERCHANT COPY		

4.4.2. TrOCR (not suitable)

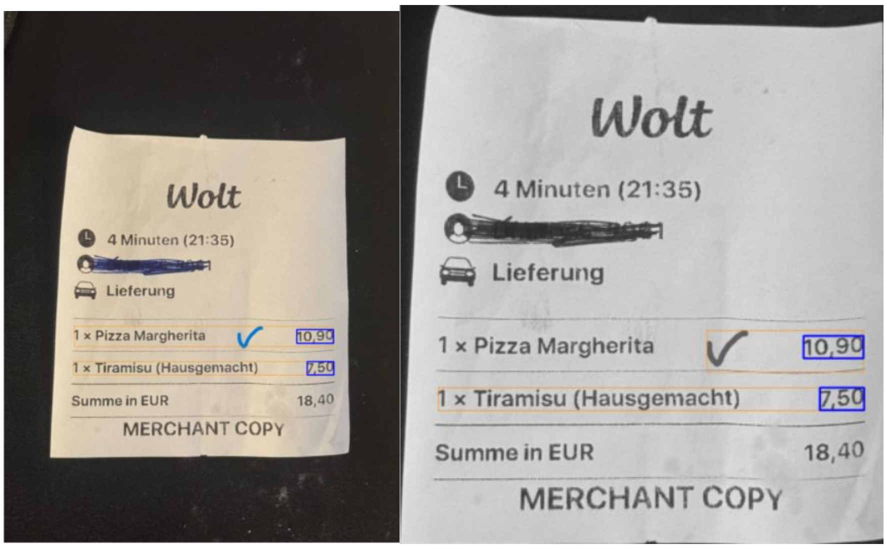
TrOCR provides high-accuracy text recognition and is robust to noise and various text types, including handwriting. Although, it outputs only flat, unstructured text, it can be effectively used for validating extracted data or for filling in missing information when other models fail to recognize certain elements.

```
trocr_processor = TrOCRProcessor.from_pretrained("microsoft/trocr-large-printed")

trocr_model = VisionEncoderDecoderModel.from_pretrained("microsoft/trocr-large-printed")

def process_receipt_trocr(image_path):

    image = Image.open(image_path).convert("RGB")

    pixel_values = trocr_processor(images=image, return_tensors="pt").pixel_values.to(device)

    generated_ids = trocr_model.generate(pixel_values)

    text = trocr_processor.batch_decode(generated_ids, skip_special_tokens=True)[0]

    return text

trocr_text = process_receipt_trocr(receipt_path)

print("TrOCR output:\n", trocr_text)
```
Listing 10. TrOCR initialization + extraction function

In this project, TrOCR did not produce any usable results for receipt images: the model either failed to recognize the receipts or returned empty outputs. Without fine-tuning on receipt data, TrOCR struggles with real-world noise. Similar to Donut it requires additional training to be effective.

4.4.3. Tesseract (approved)

Tesseract is one of the oldest and most established open-source OCR engines. While it does not natively support complex layout analysis, it remains a solid baseline solution for many real-world administrative document processing tasks.


Wolt
4 Minuten (21:35)
[redacted]
Lieferung
1 × Pizza Margherita 10,90
1 × Tiramisu (Hausgemacht) 7,50
Summe in EUR 18,40
MERCHANT COPY

Wolt
4 Minuten (21:35)
[redacted]
Lieferung
1 × Pizza Margherita 10,90
1 × Tiramisu (Hausgemacht) 7,50
Summe in EUR 18,40
MERCHANT COPY

Figure 9. Tesseract line extraction on raw and preprocessed images

After the raw lines are extracted, regular expressions (pattern matching) are applied to identify which part of each line represents the item name, the quantity, and the price. This post-processing step enables the creation of structured tables from unstructured OCR output.

Table 8. Tesseract extraction results

Source	Item	Count	Price	bbox
Raw	Pizza Margherita Y	1	10,90	[704, 768, 792, 801]
Raw	Tiramisu (Hausgemacht)	1	7,50	[729, 845, 793, 877]
Preprocessed	v		10,90	[593, 489, 681, 522]
Preprocessed	Tiramisu (Hausgemacht)	1	7,50	[617, 565, 682, 598]

Despite being one of the oldest OCR systems, Tesseract performed well in this project. It correctly detected the price for each item and determined their bounding boxes. A notable drawback is that checkmarks are recognized as the characters "V" or "Y", which can potentially be improved in the future. Additionally, it was observed that raw receipt images are recognized more accurately than preprocessed.

4.4.4. PaddleOCR (not suitable)

PaddleOCR is a modern open-source OCR toolkit based on deep learning. It supports multiple languages and has flexible configuration options for detection and recognition. In most cases, PaddleOCR can be used “out of the box” with minimal adjustment.

```
ocr = PaddleOCR(lang='de', use_angle_cls=False)
paddleocr_result_all = ocr.predict(image_path)
ocr_result_dict = paddleocr_result_all[0]
texts = ocr_result_dict.get('rec_texts', [])
bboxes = ocr_result_dict.get('rec_boxes', [])
confidences = ocr_result_dict.get('rec_scores', [])
ocr_raw_result = []
for bbox, text, conf in zip(bboxes, texts, confidences):
    ocr_raw_result.append([bbox, [text, conf]])

paddleocr_result = filter_paddleocr_result(ocr_raw_result)
df = paddleocr_to_dataframe(paddleocr_result)
```
Listing 11. PaddleOCR initialization + extraction

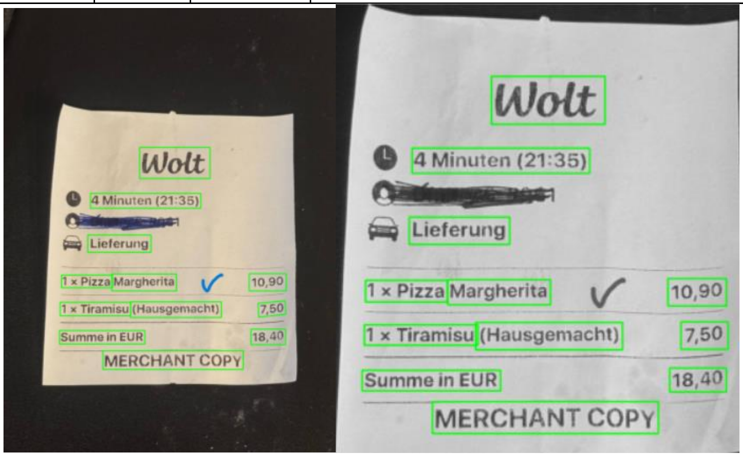
In this project, PaddleOCR did not work as expected out of the box and produced empty results for receipt images. Further improvements may be possible in the future by adjusting settings or fine-tuning the model on receipt-specific data.

4.4.5. EasyOCR (approved)

EasyOCR works well with real-world images and various languages. It is convenient for extracting receipt items and their position. Structure extraction is possible through post-processing using bounding boxes.

Table 9. EasyOCR extraction results

Source	Item	Cnt	Price	Bbox price
Raw	Pizza	1	10.90	[[696, 760], [801, 760], [801, 809], [696, 809]]
Raw	Tiramisu	1	18.40	[[701, 917], [799, 917], [799, 961], [701, 961]]
Prepr.	Pizza	1	10.90	[[586, 482], [689, 482], [689, 530], [586, 530]]
Prepr.	Tiramisu	1	7.50	[[609, 558], [692, 558], [692, 606], [609, 606]]



Wolt

4 Minuten (21:35)

[redacted]

Lieferung

1 x Pizza Margherita	10,90
1 x Tiramisu (Hausgemacht)	7,50
Summe in EUR	18,40

MERCHANT COPY

Figure 10. EasyOCR extraction on raw and preprocessed images

EasyOCR performed great on preprocessed image and had a price mistake on raw image. Another minor issue was that product names were split into parts.

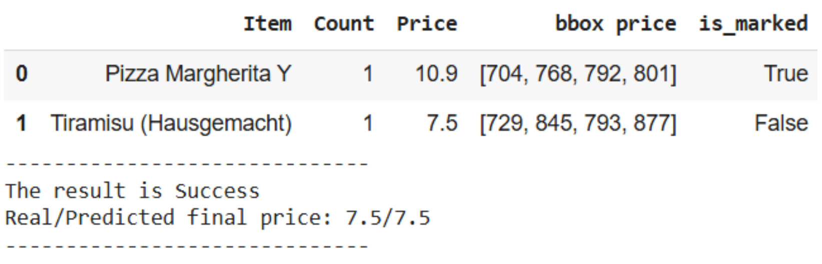
4.4.6. Conclusion

A combination of Tesseract and EasyOCR is sufficient to achieve reliable text extraction from receipts at the initial stage of this project. The pipeline can be further enhanced by integrating additional OCR models.

Link to the source code can be found at Appendix: Code Access and Source Examples

4.5. General processing pipeline

The processing pipeline for automated administrative document recognition consists of several main stages, each responsible for a specific part of image analysis and data extraction. The aim is to convert a raw photo of a paper receipt including handwritten marks into a structured digital table.



	Item	Count	Price	bbox price	is_marked
0	Pizza Margherita Y	1	10.9	[704, 768, 792, 801]	True
1	Tiramisu (Hausgemacht)	1	7.5	[729, 845, 793, 877]	False

```
------------------------------
The result is Success
Real/Predicted final price: 7.5/7.5
------------------------------
```

Figure 11. Pipeline results

```
def process_receipt(receipt_url, sum_full, sum_final, debug):
    get_raw_receipt(receipt_url, image_path, show=debug)
    success, df = ocr_extract_data(image_path, sum_full, debug)
    if not success:
        print("Raw image use failed, trying preprcossed")
        get_preprocessed_receipt(receipt_url, image_path, show=debug)
        success, df = ocr_extract_data(image_path, sum_full, debug)
    if not success:
        print(f"Failed to extract data from receipt: {receipt_url}")
        return
    print(f"Data successfully extracted")
    yolo_results = yolo_model.predict(image_path)
    if debug:
        show_yolo_results(yolo_results)
        show_checkmark_boxes(image_path, yolo_results)

    df = match_checkmarks_to_items(df, yolo_results)
    display(df)
    filtered = df[df['is_marked'] == False]
    total = filtered['Price'].astype(float).sum()
    success = total == sum_final
    print(f'The result is {"Success" if success else "Fail"}')
    print(f'Real/Predicted final price: {sum_final}/{total}')
```

Listing 12. Receipt recognition pipeline

This modular pipeline addresses all critical steps: from image acquisition and enhancement, through text and check mark extraction, to the final integration of data. Each step can be improved or adapted independently.

4.5.1. Text Extraction

At this initial stage, the pipeline receives the raw image of the receipt (using the function ***get_raw_receipt****).* The text extraction process is performed using two different OCR engines: Tesseract and EasyOCR (via the function ***ocr_extract_data***).

The extracted table of items and their prices is then validated by comparing the calculated sum of all positions with the reference sum provided on the receipt.

- If the sum matches, the process moves to the next stage.
- If it does not match, the pipeline automatically tries to process a preprocessed version of the image (using ***get_preprocessed_receipt***).
- If validation fails on both raw and processed images, the pipeline stops, as further calculations would be unreliable.

This strict validation ensures only accurate data is passed to the next steps.

4.5.2. Checkmarks Detection

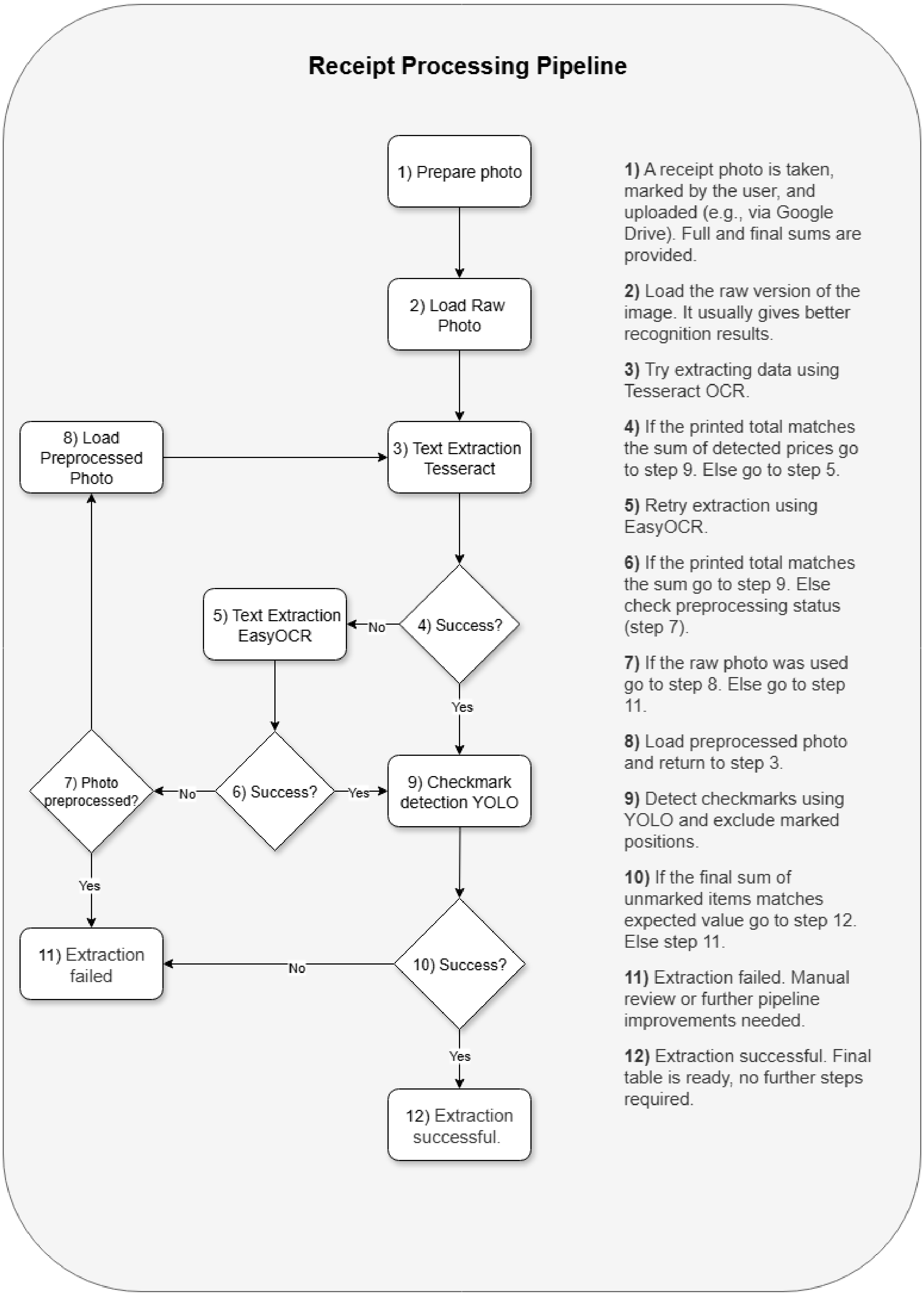
Once the table of items is extracted, the pipeline proceeds to checkmark detection. Here, a YOLO-based model is used to identify checkmarks present on the receipt (using pretrained yolo model with function ***yolo_model.predict***). The detection is performed on the same image (raw or preprocessed) that passed the previous validation stage, as YOLO produces reliable results on both image types.

After detection, the function ***match_checkmarks_to_items*** is called to integrate the checkmark locations with the recognized items in the table. Each item is flagged if it is marked by a checkmark.

4.5.3. Combining Results and Final Validation

In the final stage, the marked items are filtered out from the table to produce the list of items actually considered as expenses. The pipeline then calculates the total sum for the remaining (unmarked) items. For test scenarios calculated final price is compared with the real value. If the sum matches, the results are considered correct.

Link to the source code can be found at Appendix: Code Access and Source Examples


Receipt Processing Pipeline
1) Prepare photo
2) Load Raw Photo
3) Text Extraction Tesseract
8) Load Preprocessed Photo
4) Success?
No
5) Text Extraction EasyOCR
Yes
6) Success?
No
7) Photo preprocessed?
Yes
9) Checkmark detection YOLO
Yes
11) Extraction failed
No
10) Success?
Yes
12) Extraction successful.

1) A receipt photo is taken, marked by the user, and uploaded (e.g., via Google Drive). Full and final sums are provided.

2) Load the raw version of the image. It usually gives better recognition results.

3) Try extracting data using Tesseract OCR.

4) If the printed total matches the sum of detected prices go to step 9. Else go to step 5.

5) Retry extraction using EasyOCR.

6) If the printed total matches the sum go to step 9. Else check preprocessing status (step 7).

7) If the raw photo was used go to step 8. Else go to step 11.

8) Load preprocessed photo and return to step 3.

9) Detect checkmarks using YOLO and exclude marked positions.

10) If the final sum of unmarked items matches expected value go to step 12. Else step 11.

11) Extraction failed. Manual review or further pipeline improvements needed.

12) Extraction successful. Final table is ready, no further steps required.

Figure 12. General processing pipeline scheme

4.6. Quality assessment (metrics, comparison)

To make sure that the automated document processing pipeline worked well and gave correct results, I used several clear metrics and checking methods at each stage. The evaluation process was designed to show both how well each part of the system worked (OCR, checkmark detection) and how good the final table of results was. This helped me to choose the right models and compare them fairly.

4.6.1. Text Extraction (OCR)

Accuracy of text recognition: How many item names and prices were correctly read, compared to the real (manually marked) receipts.

Total sum check: The pipeline summed up all the detected prices and compared this sum to the sum printed on the receipt. If the sums matched, the extraction was called successful. This helped to catch any mistakes right away.

4.6.2. Checkmark Detection

Precision and recall: How many checkmarks were correctly found, and how many were missed or wrongly detected, compared to the real (annotated) data. This was checked on both synthetic and real receipts.

Mean Average Precision (mAP): The average precision for different overlap thresholds (like mAP@0.5). This was used to compare YOLO and other models (like Faster R-CNN or SSD).

4.6.3. Full Pipeline

Final sum check: How often the final table gave the same sum as the expected.

End-to-end success rate: The percent of receipts that were fully and correctly processed, which showed how reliable the pipeline was in real conditions.

Error analysis: A short summary of the most common mistakes (like wrong OCR, missed checkmarks) and how they affected the final result.

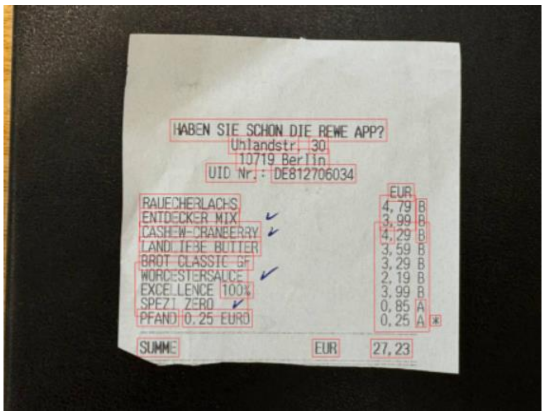
4.6.4. Summary

All these metrics were calculated on validation and test datasets, which include synthetic and real images for checkmark detection and real receipts for full pipeline testing. Tables and figures that illustrate the pipeline’s quality and performance are presented in the Results section.

5. Difficulties and Solutions found

Most of the challenges were related to price extraction, as even a single error in this part could break the entire pipeline.

5.1. Incorrect Vertical boxing

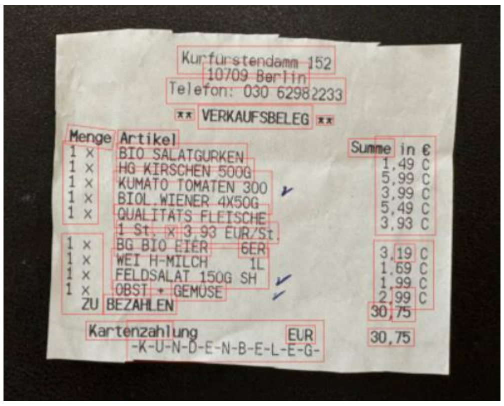


```
HABEN SIE SCHON DIE REWE APP?
       Uhlandstr. 30
       10719 Berlin
   UID Nr.: DE812706034
                                EUR
RAUECHERLACHS                   4,79 B
ENTDECKER MIX                   3,99 B
CASHEW-CRANBERRY                4,29 B
LANDLIEBE BUTTER                3,59 B
BROT CLASSIC GF                 3,29 B
WORCESTERSAUCE                  2,19 B
EXCELLENCE 100%                 3,99 B
SPEZI ZERO                      0,85 A
PFAND 0,25 EURO                 0,25 A *

SUMME                  EUR     27,23
```

Figure 13. Vertical boxing example (Tesseract)

Some receipts have tightly aligned price columns, which caused Tesseract to incorrectly merge multiple prices from different lines into a single text block. This happened due to its default layout assumptions. The issue was resolved by setting *config='--psm 6'*, which tells Tesseract to treat the image as a single uniform block of text and improves line separation.

As for EasyOCR it does not allow layout configuration like Tesseract. To handle this, a custom parsing approach was used. After running readtext(), each word's center point was calculated and stored. This sorting allowed reconstruction of the receipt content in proper reading order. This improved the results in some cases but for some receipts the problem remains actual.

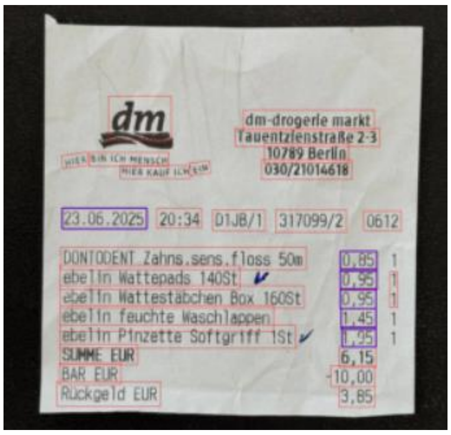


```
          Kurfürstendamm 152
            10709 Berlin
       Telefon: 030 62982233
         ** VERKAUFSBELEG **
Menge Artikel                        Summe in €
1 x   BIO SALATGURKEN                     1,49 C
1 x   HG KIRSCHEN 500G                    5,99 C
1 x   KUMATO TOMATEN 300                  3,99 C
1 x   BIOL.WIENER 4X50G                   5,49 C
1 x   QUALITÄTS FLEISCHE                  3,93 C
      1 St. x 3,93 EUR/St.
1 x   BG BIO EIER      6ER                3,19 C
1 x   WEI H-MILCH       1L                1,69 C
1 x   FELDSALAT 150G SH                   1,99 C
1 x   OBST + GEMÜSE                       2,99 C
   ZU BEZAHLEN                           30,75

  Kartenzahlung                 EUR      30,75
      -K-U-N-D-E-N-B-E-L-E-G-
```

Figure 14. Vertical boxing example (EasyOCR after)

5.2. Date False Parsing

Some parts of the receipt, such as dates (e.g., 23.06), can mistakenly be interpreted as additional costs. This can break the pipeline by changing the total sum. To prevent this, a specific regular expression was used to detect and exclude dates from price extraction: *re.fullmatch(r"\d{2}[.]\d{2}[.]\d{2,4}", p)*



dm

HIER BIN ICH MENSCH HIER KAUF ICH EIN

dm-drogerie markt
Tauentzienstraße 2-3
10789 Berlin
030/21014618

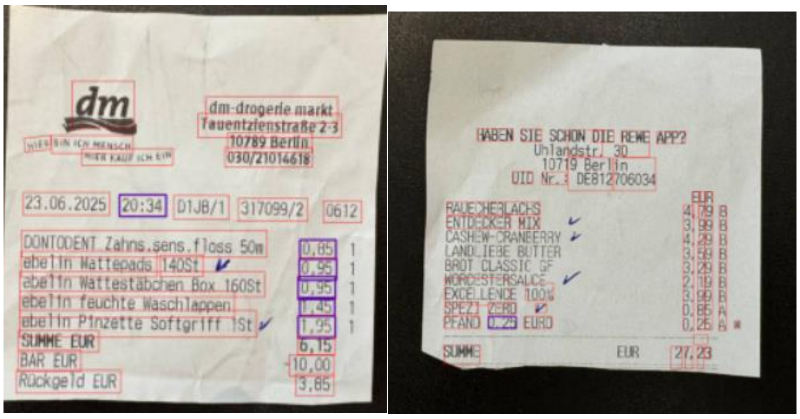
```
23.06.2025  20:34  D1JB/1  317099/2  0612
DONTODENT Zahns.sens.floss 50m      0,85  1
ebelin Wattepads 140St              0,95  1
ebelin Wattestäbchen Box 160St      0,95  1
ebelin feuchte Waschlappen          1,45  1
ebelin Pinzette Softgriff 1St       1,95  1
SUMME EUR                           6,15
BAR EUR                            10,00
Rückgeld EUR                        3,85
```

Figure 15. False price recognition (date)

5.3. Left Side Price Filtering

Sometimes the OCR engine incorrectly recognized time values like “20:34” as “20.34”, mistaking them for prices. Unfortunately, it's not possible to simply filter out all numbers with dots. Some receipts use dots as decimal separators, and OCR can also misinterpret commas as dots.

To address this specific case, a custom function *is_on_right_side* was implemented. It filters out price candidates located in the left 60% of the receipt image, as prices are on the right side of the receipt. This also helped eliminate duplicate price detections when a price appeared in the item name.



dm

HIER BIN ICH MENSCH HIER KAUF ICH EIN

dm-drogerie markt
Tauentzienstraße 2-3
10789 Berlin
030/21014618

```
23.06.2025  20:34  D1JB/1  317099/2  0612
DONTODENT Zahns.sens.floss 50m      0,85  1
ebelin Wattepads 140St              0,95  1
ebelin Wattestäbchen Box 160St      0,95  1
ebelin feuchte Waschlappen          1,45  1
ebelin Pinzette Softgriff 1St       1,95  1
SUMME EUR                           6,15
BAR EUR                            10,00
Rückgeld EUR                        3,85
```

```
HABEN SIE SCHON DIE REWE APP?
         Uhlandstr. 30
         10719 Berlin
     UID Nr.: DE812706034
                              EUR
RAEUCHERLACHS                4,79 B
ENTDECKER MIX                3,99 B
CASHEW-CRANBERRY             4,29 B
LANDLIEBE BUTTER             3,59 B
BROT CLASSIC GF              3,29 B
WORCESTERSAUCE               2,19 B
EXCELLENCE 100%              3,99 B
SPEZI ZERO                   0,85 A
PFAND 0,25 EURO              0,25 A *
SUMME              EUR      27,23
```

Figure 16. False price detection

6. Results

6.1. Full Pipeline Statistics

To understand how well the pipeline works with different types of receipts, a set of real receipts was collected and manually marked and were grouped into three levels.

Easy: 1 or 2 items. No prices inside item names and no discounts (no negative prices).
Medium: Fewer than 10 items. May include prices inside item names.
Hard: 10 or more items, or includes both prices in names and discounts

Table 10. Receipts recognition results

ID	Level	EasyOCR helped?	Preprocess helped?	Succeed?	Possible fail reason
1	Easy	N/A	N/A	Yes	
2	Easy	N/A	N/A	Yes	
3	Easy	Yes	N/A	Yes	
4	Medium	N/A	N/A	Yes	
5	Medium	Yes	N/A	Yes	
6	Medium	No	No	No	Sum keyword detection
7	Medium	N/A	N/A	Yes	
8	Medium	No	No	No	Right price recognition (OCR)
9	Hard	No	No	No	“Summe” recognition (OCR)
10	Hard	N/A	N/A	No	Missed checkmark (YOLO)
11	Hard	N/A	N/A	Yes	
12	Hard	N/A	N/A	Yes	
13	Hard	No	No	No	Price boxes detection
14	Hard	N/A	N/A	No	Checkmark position (YOLO)

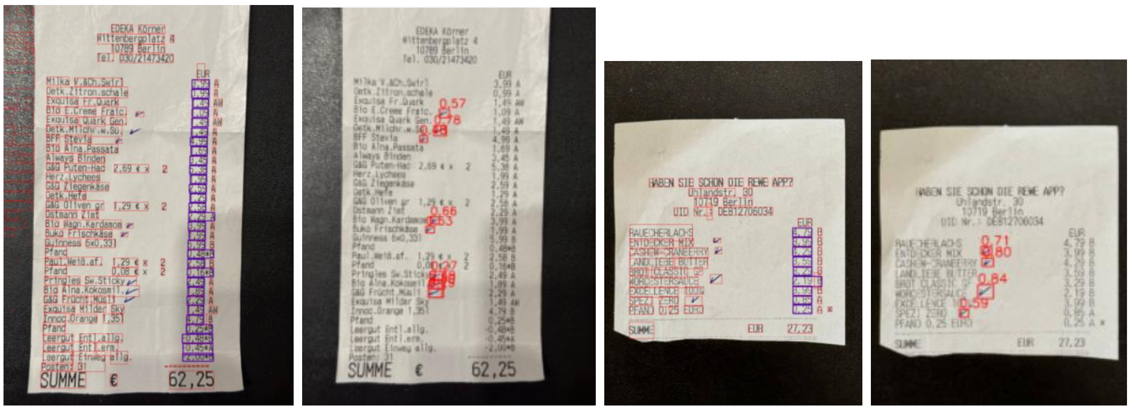
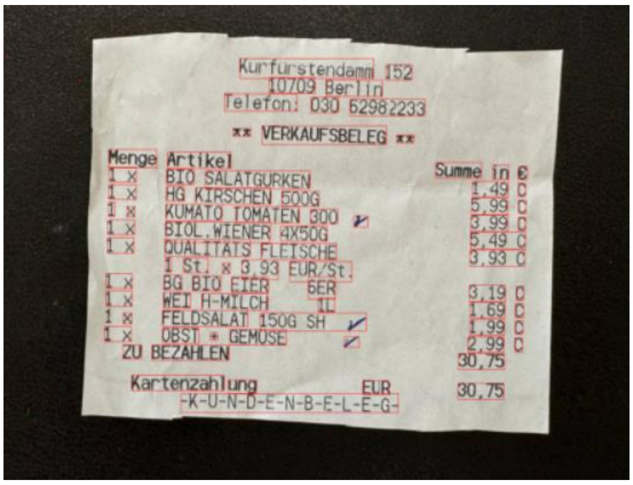


Figure 17. Correct recognition examples

6.2. Failed Recognitions Overview

6.2.1. Receipt № 6



Kurfürstendamm 152
10709 Berlin
Telefon: 030 62982233
** VERKAUFSBELEG **

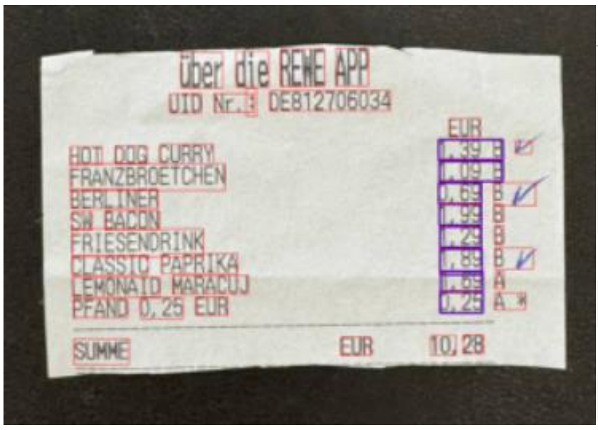
Menge	Artikel	Summe in €
1 x	BIO SALATGURKEN	1,49 C
1 x	HG KIRSCHEN 500G	5,99 C
1 x	KUMATO TOMATEN 300	3,99 C
1 x	BIOL.WIENER 4X50G	5,49 C
1 x	QUALITATS FLEISCHE	3,93 C
	1 St. x 3,93 EUR/St.	
1 x	BG BIO EIER 6ER	3,19 C
1 x	WEI H-MILCH 1L	1,69 C
1 x	FELDSALAT 150G SH	1,99 C
1 x	OBST + GEMÜSE	2,99 C
	ZU BEZAHLEN	30,75

Kartenzahlung EUR 30,75
-K-U-N-D-E-N-B-E-L-E-G-

Figure 18. Receipt № 6

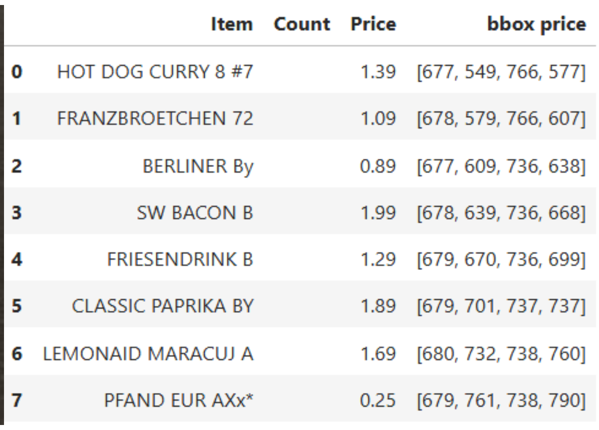
Although Tesseract managed to detect all the text-boxes correctly, the pipeline failed to recognize them as prices. The possible reason for this is that the "Summe" keyword is in the top of the receipt which blocks pipeline from further search of the prices.

6.2.2. Receipt № 8



Über die REWE APP
UID Nr.: DE812706034

	EUR
HOT DOG CURRY	1,39 B
FRANZBROETCHEN	1,09 B
BERLINER	0,69 B
SW BACON	1,99 B
FRIESENDRINK	1,29 B
CLASSIC PAPRIKA	1,89 B
LEMONAID MARACUJ	1,69 A
PFAND 0,25 EUR	0,25 A *
SUMME EUR	10,28

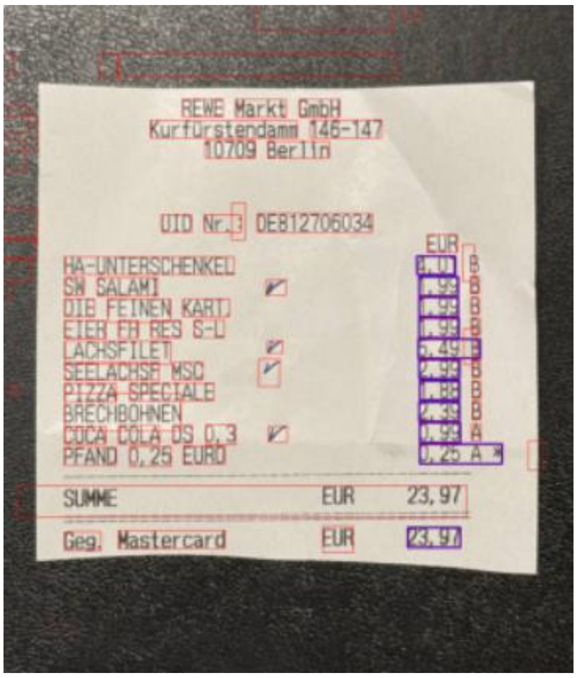


	Item	Count	Price	bbox price
0	HOT DOG CURRY 8 #7		1.39	[677, 549, 766, 577]
1	FRANZBROETCHEN 72		1.09	[678, 579, 766, 607]
2	BERLINER By		0.89	[677, 609, 736, 638]
3	SW BACON B		1.99	[678, 639, 736, 668]
4	FRIESENDRINK B		1.29	[679, 670, 736, 699]
5	CLASSIC PAPRIKA BY		1.89	[679, 701, 737, 737]
6	LEMONAID MARACUJ A		1.69	[680, 732, 738, 760]
7	PFAND EUR AXx*		0.25	[679, 761, 738, 790]

Figure 19. Receipt № 8

While all text and price bounding boxes were successfully detected, the price of "Berliner" was incorrectly recognized as €0.89 instead of the correct value €0.69.

6.2.3. Receipt № 9

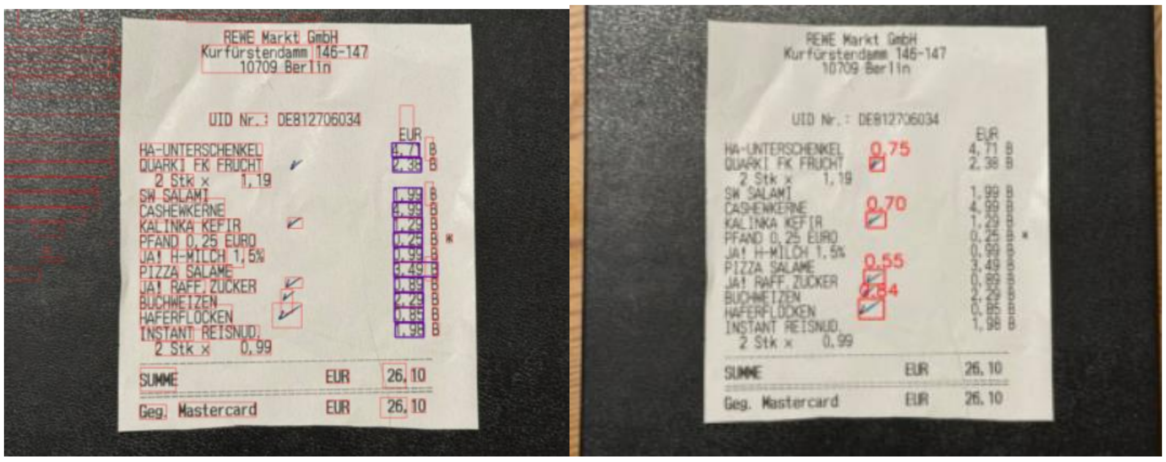


```
        REWE Markt GmbH
     Kurfürstendamm 146-147
         10709 Berlin


     UID Nr.: DE812706034
                                 EUR
HA-UNTERSCHENKEL          [illegible] B
SW SALAMI                       1,99 B
DIE FEINEN KART                 1,99 B
EIER FR RES S-U                 1,99 B
LACHSFILET                      6,49 B
SEELACHSF MSC                   2,99 B
PIZZA SPECIALE                  1,88 B
BRECHBOHNEN                     2,39 B
COCA COLA DS 0,3                0,99 A
PFAND 0,25 EURO                 0,25 A *
--------------------------------------
SUMME                  EUR     23,97
======================================
Geg. Mastercard        EUR     23,97
```

Figure 20. Receipt № 9

In this case, the extraction of the keyword “SUMME” failed. Tesseract misrecognized it as “SUMHE”, which prevented the pipeline from correctly identifying the end of the item list, resulting in the inclusion of subsequent prices as regular items.

6.2.4. Receipt № 10



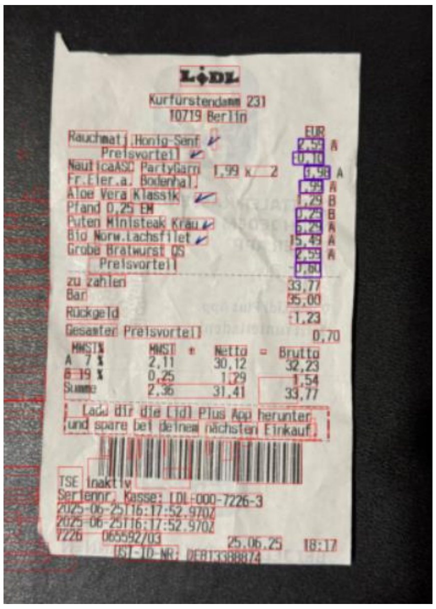
0,75
0,70
0,55
0,84

```
        REWE Markt GmbH
     Kurfürstendamm 146-147
         10709 Berlin


     UID Nr.: DE812706034
                                 EUR
HA-UNTERSCHENKEL                4,71 B
QUARKI FK FRUCHT                2,38 B
  2 Stk x      1,19
SW SALAMI                       1,99 B
CASHEWKERNE                     4,99 B
KALINKA KEFIR                   1,29 B
PFAND 0,25 EURO                 0,25 B *
JA! H-MILCH 1,5%                0,99 B
PIZZA SALAME                    3,49 B
JA! RAFF.ZUCKER                 0,89 B
BUCHWEIZEN                      2,29 B
HAFERFLOCKEN                    0,85 B
INSTANT REISNUD.                1,98 B
  2 Stk x      0,99
--------------------------------------
SUMME                  EUR     26,10
======================================
Geg. Mastercard        EUR     26,10
```

Figure 21. Receipt № 10

All bounding boxes were successfully detected and all prices were correctly extracted. This time YOLO failed to recognize one checkmark (the smallest one).

6.2.5. Receipt № 13

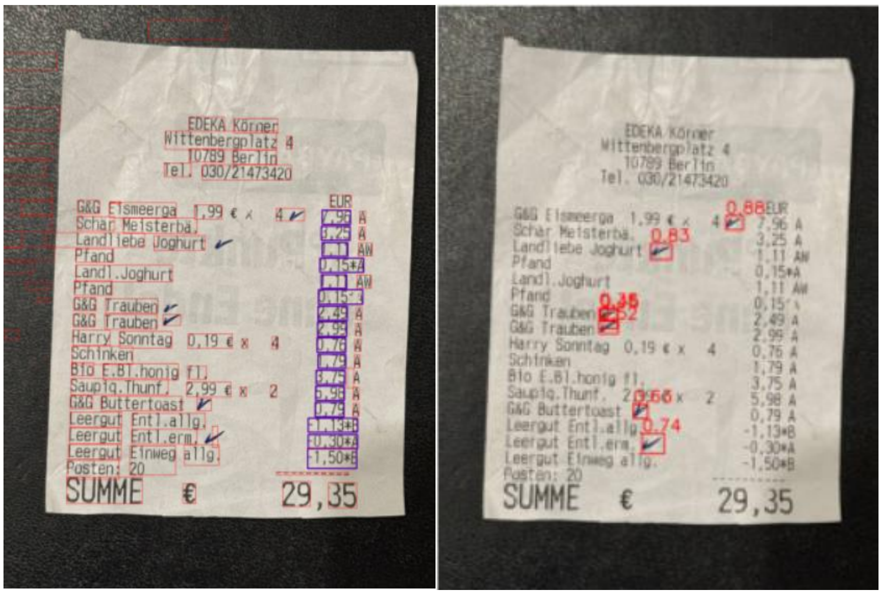


```
LIDL
Kurfürstendamm 231
10719 Berlin
                                          EUR
Rauchmati Honig-Senf                     2,59 A
   Preisvorteil                         -0,10
NauticaASC PartyGarn  1,99 x  2          3,98 A
Fr.Eier.a. Bodenhal                       ,99 A
Aloe Vera Klassik                        1,29 B
Pfand 0,25 EM                            0,25 B
Puten Ministeak Kräu                     6,29 A
Bio Norw.Lachsfilet                     15,49 A
Grobe Bratwurst QS                       2,59 A
   Preisvorteil                         -0,60
-----------------------------------------------
zu zahlen                               33,77
Bar                                     35,00
Rückgeld                                -1,23
Gesamter Preisvorteil                    0,70
 MWST%     MWST  +  Netto  =  Brutto
A  7 %     2,11     30,12      32,23
B 19 %     0,25      1,29       1,54
Summe      2,36     31,41      33,77
-----------------------------------------------
   Lade dir die Lidl Plus App herunter
und spare bei deinem nächsten Einkauf.
TSE Inaktiv
Seriennr. Kasse: LDL-000-7226-3
2025-06-25T16:17:52.970Z
2025-06-25T16:17:52.970Z
7226   065592/03          25.06.25    18:17
      UST-ID-NR: DE813388874
```

Figure 22. Receipt № 13

Two prices were not recognized by the pipeline, possibly due to inaccuracies in OCR extraction or failures during the post-processing of detected prices.

6.2.6. Receipt № 14



```
          EDEKA Körner
       Wittenbergplatz 4
         10789 Berlin
       Tel. 030/21473420
                                      EUR
G&G Eismeerga   1,99 € x    4        7,96 A
Schär Meisterbä.                     3,25 A
Landliebe Joghurt                    1,11 AW
Pfand                                0,15*A
Landl.Joghurt                        1,11 AW
Pfand                                0,15*A
G&G Trauben                          2,49 A
G&G Trauben                          2,99 A
Harry Sonntag   0,19 € x    4        0,76 A
Schinken                             1,79 A
Bio E.Bl.honig fl.                   3,75 A
Saupiq.Thunf.   2,99 € x    2        5,98 A
G&G Buttertoast                      0,79 A
Leergut Entl.allg.                  -1,13*B
Leergut Entl.erm.                   -0,30*A
Leergut Einweg allg.                -1,50*B
Posten: 20                     -----------
SUMME        €                      29,35
```

Figure 23. Receipt № 14

Text and price extraction were successful, and all checkmarks were correctly detected. However, the pipeline failed to associate the checkmark position with the corresponding price, resulting in the omission of the €2.99 item from the final output.

Debug output: *The result is Fail. Real/Predicted final price: 14.31/17.3*

6.3. Summary

A total of 14 real-world receipts were tested to assess the robustness and reliability of the receipt processing pipeline. The evaluation results are summarized as follows:

- Successful extraction was achieved for 8 out of 14 receipts (57%).
- EasyOCR successfully resolved two cases in which Tesseract failed to produce correct results.
- Preprocessing techniques were not required in any of the successfully processed cases.

The results by receipts complexity are:

Easy: 100% (3 cases)

Medium: 60% (5 cases)

Hard: 33% (6 cases)

The pipeline demonstrated high accuracy on simple receipts, but performance declined with increased complexity. Even though effectiveness of recognition was only 57% the problems of the remaining 43% are understandable and can be fixed with further improvements improving the accuracy for this set of images to 86-100%. (watch Further Work chapter).

7. Future work

7.1. Fixes for Used Set

7.1.1. Resolution Increase

All pictures used in testing were 960×1280 pixels, which is sufficient for small receipts but may be inadequate for larger ones, potentially leading to text and checkmark misrecognitions. In my opinion, using a resolution of 1536×2048 or higher (especially since automatic resizing to 2048 is already implemented) is a strong choice. This can significantly improve recognition quality, and for selected cases, such resolution will not have a major impact on processing time or storage space.

Receipt possible fixes: № 8, № 9, № 10, № 13

7.1.2. Sum Logic Update

Right now, the pipeline stops searching for prices after the “sum” keyword is found. In some receipts, this keyword appears before the prices or is not correctly recognized by OCR. Since the total sum is provided along with the photo, additional strategies could be added, such as comparing each price.

Receipt possible fixes: № 6, № 9

7.1.3. More OCR Models

As there are no strict time constraints for receipt recognition, it is can be useful to add one or two fallback models in case both Tesseract and EasyOCR fail. TrOCR appears to be the best candidate, as it delivers high accuracy and produces output that is easy to integrate.

Receipt possible fixes: № 8, № 13

7.1.4. YOLO Updates

YOLO was the most convenient and effective tool used in this thesis. With a mix of small handmade and generated datasets, it produced only one false negative and had just one issue with bounding box alignment. In both cases, the checkmarks were poorly drawn and could be fixed by applying stricter rules for how checkmarks are written. It would also be beneficial to expand the dataset more different checkmarks.

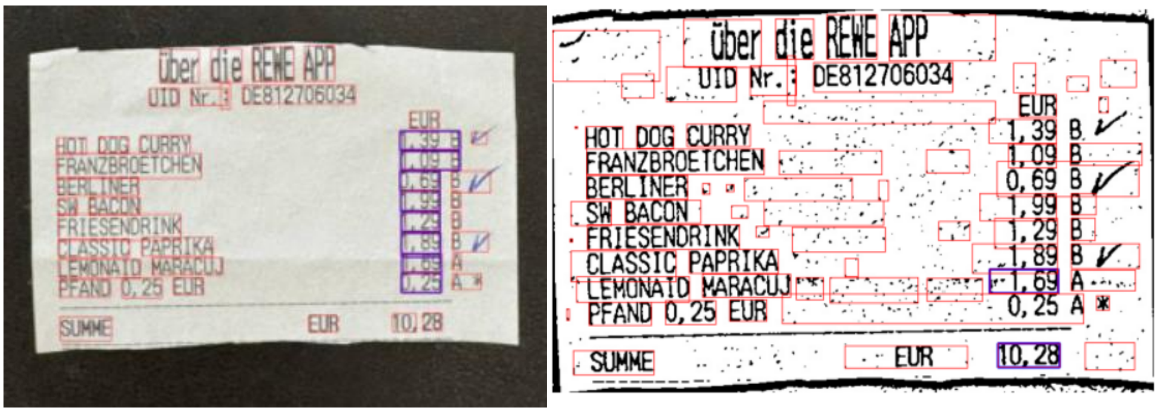
Receipt possible fixes: № 10, № 14

7.2. More Improvement Ideas

Except fixes suggested in previous chapter there are some ideas that can also improve the quality of the final result.

7.2.1. Preprocessing Strategy Change

Although I followed the recommended best practices, image preprocessing in the collected dataset led to zero improvements in detection. In fact, OCR performance worsened after preprocessing, resulting in many false positives.



über die REWE APP
UID Nr.: DE812706034

	EUR	
HOT DOG CURRY	1,39	B
FRANZBROETCHEN	1,09	B
BERLINER	0,69	B
SW BACON	1,99	B
FRIESENDRINK	1,29	B
CLASSIC PAPRIKA	1,89	B
LEMONAID MARACUJ	1,69	A
PFAND 0,25 EUR	0,25	A *
SUMME EUR	10,28	

Figure 24. Raw/Preprocessed OCR output

It is possible that preprocessing may yield better results with further refinement or when used with different OCR models. Moreover, in the current pipeline, preprocessing is skipped for YOLO if OCR succeeds on the raw image, due to cropping issues. Adjusting the pipeline to address this could potentially improve YOLO's performance as well.

7.2.2. Item Name Parsing Enhancements

In the current pipeline, the item name is defined as everything in the text line except the price. While this works in most cases, it can occasionally result in messy or inconsistent naming. Further improvements in text parsing and structure extraction are needed to ensure cleaner item names.

7.2.3. Use of LLMs for Fallback Recognition

Modern LLMs such as ChatGPT and DeepSeek have shown strong capabilities in extracting information from receipts. They can potentially enhance various parts of the pipeline. However, their responses can be unstable and unpredictable, so they should be considered only as a fallback or last-resort option.

8. Appendix: Code Access and Source Examples

8.1. Appendix A. Reproducibility and Public Access

This section provides full access information for all digital components of the project: source code, interactive notebooks, and datasets. The following table lists all digital resources used for the preparation of the full pipeline:

Table 11. Preparation notebooks and datasets

Item	Type	Access Link
Checkmarks handmade dataset	Kaggle Dataset	https://www.kaggle.com/datasets/arka2711/checkmark-handmade
Checkmark full dataset creation	Kaggle Notebook	https://www.kaggle.com/code/arka2711/checkmarks-dataset-creation
Checkmarks full dataset	Kaggle Dataset	https://www.kaggle.com/datasets/arka2711/checkmarks-mix
Recognition models tryouts	Kaggle Notebook	https://www.kaggle.com/code/arka2711/checkmark-models
OCR models tryouts	Kaggle Notebook	https://www.kaggle.com/code/arka2711/ocr-mix

The following table lists all components used in the final pipeline. Final pipeline uses “Receipt recognition utils” dataset which includes image provider and data extraction modules along with weights for YOLO model.

Table 12. Final pipeline notebooks and dataset

Item	Type	Access Link
Final pipeline	Kaggle Notebook	https://www.kaggle.com/code/arka2711/receipt-recognition-pipeline
Receipt recognition utils	Kaggle Dataset	https://www.kaggle.com/datasets/arka2711/receipt-recognition-utils
Image Provider	Kaggle Notebook	https://www.kaggle.com/code/arka2711/image-provider
Receipt data extraction	Kaggle Notebook	https://www.kaggle.com/code/arka2711/receipt-data-extraction

All receipts used in the diploma project are available at:

https://drive.google.com/drive/folders/1nSiNLyIpZXiEUjocJ3ZL-Plcp6YwbOnw?usp=drive_link

8.2. Appendix B. Final Pipeline Source Code

This section contains the core implementation scripts of the receipt recognition pipeline. The code is separated into three main parts.

8.2.1. Main Controller Script

This is the main entry point that coordinates all pipeline steps from image input to final data validation.

8.2.1.1. Setup

Before executing the receipt recognition pipeline initial setup is required. The system installs the German language pack for Tesseract OCR (unfortunately it is impossible to implement inside the dataset) and the Ultralytics library for YOLOv8. After that, the necessary helper functions are imported from a shared utility dataset hosted on Kaggle as well as libraries for data visualization. Finally, the pretrained YOLO model for detecting checkmarks is loaded and ready to use.

```
!apt-get install -y -qq tesseract-ocr-deu # needed for Tesseract in
receipt_data_extraction
!pip install -q ultralytics


import sys
sys.path.append('/kaggle/input/receipt-recognition-utils')
from image_provider import get_raw_receipt, get_preprocessed_receipt
from receipt_data_extraction import ocr_extract_data
from ultralytics import YOLO
import cv2
import matplotlib.pyplot as plt
import pandas as pd


yolo_model_path = '/kaggle/input/receipt-recognition-utils/yolo_best.pt'
image_path = "receipt.jpg"
bbox_column = 'bbox price'


yolo_model = YOLO(yolo_model_path)
```

Listing 13. Environment setup and module initialization

8.2.1.2. Auxiliary functions

These functions are used to show YOLO model results (show_yolo_results), visualize checkmarks bounding boxes(show_checkmark_boxes) and match each predicted checkmark with a corresponding receipt item (match_checkmarks_to_items).

```
def show_yolo_results(results):
    for r in results:
        boxes = r.boxes
        for box in boxes:
            x1, y1, x2, y2 = box.xyxy[0]
            conf = box.conf[0]
            cls = int(box.cls[0])
            print(f"Detected checkmark: class={cls}, conf={conf:.2f},
bbox=({x1:.0f}, {y1:.0f}, {x2:.0f}, {y2:.0f})")
def match_checkmarks_to_items(df, yolo_results, bbox_column='bbox
price'):
    checkmarks = []
    for r in yolo_results:
        for box in r.boxes:
            x1, y1, x2, y2 = [float(v) for v in box.xyxy[0]]
            checkmarks.append([x1, y1, x2, y2])

    marks = []
    for idx, row in df.iterrows():
        bbox = row.get(bbox_column)
        if bbox is None or (isinstance(bbox, float) and np.isnan(bbox))
or (isinstance(bbox, list) and len(bbox) != 4):
            marks.append(False)
            continue
        x1, y1, x2, y2 = [float(v) for v in bbox]
        found = False
        for bx1, by1, bx2, by2 in checkmarks:
            center_chk_y = (by1 + by2) / 2
            if y1 <= center_chk_y <= y2:
                found = True
                break
        marks.append(found)
    df['is_marked'] = marks
    return df
```

```
def show_checkmark_boxes(image_path, yolo_results, color=(0, 0, 255),
thickness=2):
    img = cv2.imread(image_path)
    for r in yolo_results:
        for box in r.boxes:
            x1, y1, x2, y2 = [int(coord) for coord in box.xyxy[0]]
            cv2.rectangle(img, (x1, y1), (x2, y2), color, thickness)
            label = f'{box.conf[0]:.2f}'
            cv2.putText(img, label, (x1, y1 - 5),
cv2.FONT_HERSHEY_SIMPLEX, 1.2, color, 2)
    plt.figure(figsize=(4, 4))
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.show()
```
Listing 14. Auxiliary functions

8.2.1.3. Pipeline Testing Loop:

This block demonstrates how the pipeline is tested on different sets of receipts. Each test case includes a URL and expected values for the full and final price.

```
easy_checks = [
("https://drive.google.com/uc?export=download&id=1u7Z7FsMtSPZiXsYVPw3yWL1
uEQHJfODX", 18.40, 7.50),… ]
medium_checks = […]
hard_checks = […]


for download_url, price_full, price_final in medium_checks:
    print(f'Starting process for URL: {download_url}, price_full:
{price_full}, price_final: {price_final} ')
    process_receipt(download_url, price_full, price_final, True)
```
Listing 15. Pipeline testing

8.2.1.4. Receipt Recognition Function

This function represents the complete logic of the receipt recognition pipeline. It takes a receipt image URL and expected totals as input, runs OCR, performs checkmark detection using a YOLO model, and compares the extracted sum with the expected final value. It includes fallback logic, visualization for debugging, and final result output.

```python
def process_receipt(receipt_url, sum_full, sum_final, debug):
    get_raw_receipt(receipt_url, image_path, show=debug)
    success, df = ocr_extract_data(image_path, sum_full, debug)
    if not success:
        print("Raw image use failed, trying preprcessed")
        get_preprocessed_receipt(receipt_url, image_path, show=debug)
        success, df = ocr_extract_data(image_path, sum_full, debug)

    if not success:
        print(f"Failed to extract data from receipt: {receipt_url}")
        return

    print(f"Data successfully extracted")

    yolo_results = yolo_model.predict(image_path)
    if debug:
        show_yolo_results(yolo_results)
        show_checkmark_boxes(image_path, yolo_results)

    df = match_checkmarks_to_items(df, yolo_results)
    display(df)

    filtered = df[df['is_marked'] == False]
    total = round(filtered['Price'].astype(float).sum(), 2)
    success = total == sum_final
    print("------------------------------")
    print(f'The result is {"Success" if success else "Fail"}')
    print(f'Real/Predicted final price: {sum_final}/{total}')
    print("------------------------------")
```

Listing 16. Receipt Recognition Function

8.2.2. Image provider

This module is responsible for downloading, resizing, and preprocessing receipt images. It provides a consistent interface for working with both raw and cleaned images.

8.2.2.1. Imports, get and preprocess functions

This code includes necessary imports, test code and get/preprocess functions. These functions are used to get raw or preprocessed receipt.

```python
import cv2
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import os
import gdown

def get_raw_receipt(url, out_path, show=True):
    image = download_image(url, out_path)
    if show: show_image(image)
def get_preprocessed_receipt(url, out_path, show=True):
    download_image(url, out_path)
    image = preprocess_receipt(out_path)
    if show: show_image(image)
def preprocess_receipt(img_path):
    receipt = cv2.imread(img_path)
    receipt = cv2.cvtColor(receipt, cv2.COLOR_BGR2GRAY)
    _, receipt_otsu = cv2.threshold(receipt, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)
    contours, _ = cv2.findContours(receipt_otsu, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    cnt = max(contours, key=cv2.contourArea)
    x, y, w, h = cv2.boundingRect(cnt)
    receipt = receipt[y:y+h, x:x+w]
    receipt = cv2.adaptiveThreshold(
        receipt, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 21,
8)
    receipt = cv2.medianBlur(receipt, 3)
    cv2.imwrite(img_path, receipt)
    return receipt
```

Listing 17. Get and preprocess functions

8.2.2.2. Auxiliary functions and test code

These functions are used to download image (download_image), resize (smart_resize) and display it (show_image). Test code part demonstrates how the code is working.

```python
def smart_resize(image, max_width=2048, max_height=2048):
    height, width = image.shape[:2]
    scale = min(max_width / width, max_height / height, 1.0)
    if scale >= 1:
        print('No resize needed')
        return image
    else:
        new_width, new_height = int(width * scale), int(height * scale)
        resized = cv2.resize(image, (new_width, new_height), 
interpolation=cv2.INTER_AREA)
        print(f'Was resized. New width: {new_width}, new height: 
{new_height}')
        return resized


def show_image(img, title='Image', figsize=(5, 5)):
    if isinstance(img, str):
        image = cv2.imread(img)
        if image is None:
            print(f"Can't open: {img}")
            return
    else:
        image = img

    plt.figure(figsize=figsize)
    if len(image.shape) == 2:
        plt.imshow(image, cmap='gray')
    else:
        plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title(title)
    plt.axis('off')
    plt.show()
```

```
def download_image(url, out_path, resize=True, max_width=2048,
max_height=2048):
    gdown.download(url, out_path, quiet=False)
    if resize:
        img = cv2.imread(out_path)
        img = smart_resize(img, max_width=max_width,
max_height=max_height)
        cv2.imwrite(out_path, img)
    print(f'Image successfully loaded: {out_path}')
    return img


# Test code
# out_path = "receipt.jpg"
# receipt_url = "https://drive.google.com/uc?export=download&id=1Oib6OqD-
alWpc0Df0HDhGKqeLXXmDgjL"
# get_raw_receipt(receipt_url_default, out_path, show=True)
# get_preprocessed_receipt(receipt_url_default, out_path, show=True)
```

Listing 18. Preprocessing auxiliary functions and test code

8.2.3. Receipt Data Extraction Module

This module is responsible for extracting structured item and price data from receipt images using two OCR engines (Tesseract and EasyOCR) with fallback logic and keyword filtering. It attempts to locate prices and associate them with product lines based on position and content.

8.2.3.1. Imports and Auxiliary functions

Before using OCR, the script includes a setup block that imports key libraries and prepares helper functions for price filtering (get_prices / is_on_the_rigth_side) and bbox conversion (easyocr_bbox_to_xyxy) along with the set of keywords needed for sum recognition. The commented-out command for installing the German language pack for Tesseract is necessary but remains commented out since Kaggle does not allow system-level package installations.

```
#!apt-get install -y -qq tesseract-ocr-deu


import pytesseract
from PIL import Image
import pandas as pd
import re
import cv2
import numpy as np
import matplotlib.pyplot as plt
import easyocr
import torch


sum_keywords = ["summe", "total", "gesamt", "netto", "brutto", "bar", "zu
bezahlen", "zu zahlen", "zahlen"]
def get_prices(line_text):
    raw_prices = re.findall(r"-?\d+[,.]\d{2}", line_text)
    prices = []
    for p in raw_prices:
        if re.fullmatch(r"\d{2}[.]\d{2}[.]\d{2,4}", p):
            continue
        prices.append(p)
    return prices
```

```python
def is_on_right_side(bbox, image_width, threshold_ratio=0.5): # helps to skip prices in naming and dates
    x_center = (bbox[0] + bbox[2]) / 2
    return x_center >= image_width * threshold_ratio


def easyocr_bbox_to_xyxy(bbox):
    xs = [p[0] for p in bbox]
    ys = [p[1] for p in bbox]
    x1, x2 = min(xs), max(xs)
    y1, y2 = min(ys), max(ys)
    return [x1, y1, x2, y2]
```

Listing 19. OCR imports and auxiliary functions

8.2.3.2. Tesseract and EasyOCR lines extractions

These two functions (extract_lines_tesseract / extract_lines_easyocr) extract lines of text from the image.

```python
def extract_lines_tesseract(image_path):
    image = Image.open(image_path).convert("RGB")
    ocr_data = pytesseract.image_to_data(image, config='--psm 6', output_type=pytesseract.Output.DICT, lang='deu')
    rows = []
    current_line = []
    current_y = None
    for i in range(len(ocr_data['text'])):
        text = ocr_data['text'][i].strip()
        #print(f'i={i}: {text}')
        if text:
            y = ocr_data['top'][i]
            if current_y is None or abs(y - current_y) < 10:
                current_line.append((text, i))
                current_y = y
            else:
                rows.append(current_line.copy())
                current_line = [(text, i)]
                current_y = y
    if current_line:
        rows.append(current_line.copy())
    return rows, ocr_data, image
```

```python
def extract_lines_easyocr(image_path, y_threshold=10):
    reader = easyocr.Reader(['de', 'en'], gpu=torch.cuda.is_available())
    results = reader.readtext(image_path)
    word_boxes = []
    for bbox, text, conf in results:
        x_center = sum([pt[0] for pt in bbox]) / 4
        y_center = sum([pt[1] for pt in bbox]) / 4
        word_boxes.append((text, bbox, x_center, y_center))

    word_boxes.sort(key=lambda x: (x[3], x[2]))
    grouped_lines = []
    current_line = []
    last_y = None

    for text, bbox, x_center, y_center in word_boxes:
        if last_y is None or abs(y_center - last_y) < y_threshold:
            current_line.append((text, bbox))
        else:
            grouped_lines.append(current_line)
            current_line = [(text, bbox)]
        last_y = y_center

    if current_line:
        grouped_lines.append(current_line)

    return grouped_lines
```
Listing 20. OCR extract lines

8.2.3.3. Tesseract process receipt

This function returns a total sum calculated and data frame with receipt data using Tesseract. It also shows the bounding boxes of the parsed info and found prices if debug is on.

```
def process_receipt_tesseract(image_path, img_width, show_image=True):
    rows, ocr_data, image = extract_lines_tesseract(image_path)
    img_cv = cv2.cvtColor(np.array(image), cv2.COLOR_RGB2BGR)

    price_boxes = []
    items_data = []

    for i in range(len(ocr_data['text'])):
        text = ocr_data['text'][i].strip()
        if text:
            x, y, w, h = ocr_data['left'][i], ocr_data['top'][i],
ocr_data['width'][i], ocr_data['height'][i]
            cv2.rectangle(img_cv, (x, y), (x + w, y + h), (0, 255, 0), 1)
            cv2.putText(img_cv, text, (x, y - 5),
cv2.FONT_HERSHEY_SIMPLEX, 0.4, (0, 255, 0), 1)

    for line in rows:
        line_text = ' '.join([w for w, _ in line])
        line_lower = line_text.lower()

        if any(kw in line_lower for kw in sum_keywords):
            break
        prices = get_prices(line_text)
        if prices:
            price_str = prices[-1].replace(",", ".")  # last price
        else:
            price_str = None
        count_match = re.search(r"(\d+) ?x", line_text)
        name = re.sub(r"\d+[,.]\d{2}|\d+ ?x", "", line_text).strip()
        price_bbox = None
        price_val = None
        if price_str:
            for word, idx in reversed(line):  # from last price
                if price_str in word.replace(",", "."):
                    temp_bbox = [
                        ocr_data['left'][idx],
                        ocr_data['top'][idx],
                        ocr_data['left'][idx] + ocr_data['width'][idx],
                        ocr_data['top'][idx] + ocr_data['height'][idx]
                    ]
                    if is_on_right_side(temp_bbox, img_width):
                        price_bbox = temp_bbox
                        try:
                            price_val = float(price_str)
                        except Exception:
                            price_val = None
                        break
        if price_bbox:
            price_boxes.append(price_bbox)
            items_data.append({
                'Item': name,
                'Count': count_match.group(1) if count_match else '',
                'Price': price_val,
                'bbox price': price_bbox
            })
    df = pd.DataFrame(items_data)
    total_sum = 0
    if len(df) > 0:
        total_sum = df['Price'].dropna().sum()
        total_sum = round(total_sum, 2)
    if show_image:
        img_cv = cv2.cvtColor(np.array(image), cv2.COLOR_RGB2BGR)
        for box in price_boxes:
            x1, y1, x2, y2 = [int(z) for z in box]
            cv2.rectangle(img_cv, (x1, y1), (x2, y2), (255, 0, 0), 2)
        for i in range(len(ocr_data['text'])):
            text = ocr_data['text'][i].strip()
            if text:
                x, y, w, h = ocr_data['left'][i], ocr_data['top'][i],
ocr_data['width'][i], ocr_data['height'][i]
                cv2.rectangle(img_cv, (x, y), (x + w, y + h), (0, 0,
255), 1)

        plt.figure(figsize=(6, 6))
        plt.imshow(cv2.cvtColor(img_cv, cv2.COLOR_BGR2RGB))
        plt.axis('off')
        plt.show()

    return df, total_sum
```

Listing 21. Tesseract process receipt function

8.2.3.4. EasyOCR process receipt

This function returns a total sum calculated and data frame with receipt data using EasyOCR. It also shows the bounding boxes of the parsed info and found prices if debug is on.

```python
def process_receipt_easyocr(image_path, img_width, show_image):
    grouped_lines = extract_lines_easyocr(image_path)

    rows = []
    for line in grouped_lines:
        line_text = ' '.join([text for text, _ in line]).strip().lower()
        if any(kw in line_text for kw in sum_keywords):
            break

        prices = get_prices(line_text)
        price_str = prices[-1].replace(",", ".") if prices else None
        count_match = re.search(r"(\d+) ?x", line_text)
        name = re.sub(r"\d+[,.]\d{2}|\d+ ?x", "", line_text).strip()
        price_bbox = None
        price_val = None
        if price_str:
            for text, bbox in reversed(line):
                if price_str in text.replace(",", "."):
                    temp_bbox = easyocr_bbox_to_xyxy(bbox)
                    if is_on_right_side(temp_bbox, img_width):
                        price_bbox = temp_bbox
                        try:
                            price_val = float(price_str)
                        except Exception:
                            price_val = None
                        break

        if price_bbox:
            rows.append({
                'Item': name,
                'Count': count_match.group(1) if count_match else '',
                'Price': price_val,
                'bbox price': price_bbox
            })
    df = pd.DataFrame(rows)
    total_sum = 0
    if not df.empty and "Price" in df.columns:
        total_sum = df["Price"].dropna().sum()
        total_sum = round(total_sum, 2)

    if show_image:
        img = cv2.imread(image_path)

        if not df.empty and 'bbox price' in df.columns:
            for bbox in df['bbox price'].dropna():
                x1, y1, x2, y2 = [int(coord) for coord in bbox]
                cv2.rectangle(img, (x1, y1), (x2, y2), (255, 0, 0), 2)

        for line in grouped_lines:
            for text, bbox in line:
                pts = np.array(bbox, dtype=np.int32)
                cv2.polylines(img, [pts], isClosed=True, color=(0, 0,
255), thickness=1)
        plt.figure(figsize=(5, 4))
        plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        plt.axis('off')
        plt.tight_layout()
        plt.show()

    return df, total_sum
```
Listing 22. EasyOCR process receipt function

8.2.3.5. Extract data function and test code

Extract function (ocr_extract_data) is a unified entry point for OCR. It first attempts to extract data using Tesseract and compares the result to the expected total sum. If the validation fails, it switches to EasyOCR as a fallback and performs the same process. A test code is included to demonstrate how the pipeline functions are used together in practice.

```
def ocr_extract_data(image_path, sum_info, debug=False):
    print('Tesseract extraction')

    img = cv2.imread(image_path)
    img_width = img.shape[1]

    df, total_sum = process_receipt_tesseract(image_path, img_width,
debug)

    if df is None or df.empty:
        print('Result df is empty')
    else:
        if debug:
            display(df)
        success = sum_info == total_sum
        if success:
            return success, df

    print('Tesseract extraction failed, trying EasyOCR')
    df, total_sum = process_receipt_easyocr(image_path, img_width, debug)
    if df is None or df.empty:
        print('Result df is empty. EasyOCR extraction failed')
        return False, df

    if debug:
        display(df)

    success = sum_info == total_sum
    print(f'EasyOCR extraction success: {success}')
    return success, df

"""## Example"""
# import gdown
# receipt_url = "
https://drive.google.com/uc?export=download&id=1SrYa8wVoSA3aAKnirvV7qA6bn
xiSSTRv"
# out_path = "receipt.jpg"
# gdown.download(receipt_url, out_path, quiet=False)
# success, df = ocr_extract_data(out_path, 6.15, debug=True)
```
Listing 23. OCR extract data function + test code

9. Bibliography

[1] "Optical character recognition (Wiki)," *Wikipedia*. Jun. 02, 2025. Accessed: Jun. 19, 2025. [Online]. Available:

https://en.wikipedia.org/w/index.php?title=Optical_character_recognition&oldid=1293503121

[2] Wikipedia, "Optical character recognition (Wiki ru)," *Википедия*. May 21, 2025. Accessed: Jun. 19, 2025. [Online]. Available:

https://ru.wikipedia.org/w/index.php?title=%D0%9E%D0%BF%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D1%80%D0%B0%D1%81%D0%BF%D0%BE%D0%B7%D0%BD%D0%B0%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D1%81%D0%B8%D0%BC%D0%B2%D0%BE%D0%BB%D0%BE%D0%B2&oldid=145245515

[3] S. G. Dedgaonkar, A. A. Chandavale, and A. M. Sapkal, "Survey of Methods for Character Recognition," vol. 1, no. 5, 2012.

[4] N. Islam, Z. Islam, and N. Noor, "A Survey on Optical Character Recognition System," *J. Inf.*, vol. 10, no. 2, 2016.

[5] R. Smith, "An Overview of the Tesseract OCR Engine," in *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007) Vol 2*, Curitiba, Parana, Brazil: IEEE, Sep. 2007, pp. 629–633. doi: 10.1109/ICDAR.2007.4376991.

[6] M. Flores, D. Valiente, M. Alfaro, M. Fabregat-Jaén, and L. Payá, "Evaluation of Open-Source OCR Libraries for Scene Text Recognition in the Presence of Fisheye Distortion:," in *Proceedings of the 21st International Conference on Informatics in Control, Automation and Robotics*, Porto, Portugal: SCITEPRESS - Science and Technology Publications, 2024, pp. 133–140. doi: 10.5220/0012927600003822.

[7] G. Kim *et al.*, "OCR-free Document Understanding Transformer," Oct. 06, 2022, *arXiv*: arXiv:2111.15664. doi: 10.48550/arXiv.2111.15664.

[8] M. Li *et al.*, "TrOCR: Transformer-based Optical Character Recognition with Pre-trained Models," Sep. 06, 2022, *arXiv*: arXiv:2109.10282. doi: 10.48550/arXiv.2109.10282.

[9] Z.-Q. Zhao, P. Zheng, S. Xu, and X. Wu, "Object Detection with Deep Learning: A Review," Apr. 16, 2019, *arXiv*: arXiv:1807.05511. doi: 10.48550/arXiv.1807.05511.

[10] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," May 09, 2016, *arXiv*: arXiv:1506.02640. doi: 10.48550/arXiv.1506.02640.

[11] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," Jan. 06, 2016, *arXiv*: arXiv:1506.01497. doi: 10.48550/arXiv.1506.01497.

[12] W. Liu *et al.*, “SSD: Single Shot MultiBox Detector,” vol. 9905, 2016, pp. 21–37. doi: 10.1007/978-3-319-46448-0_2.

[13] “Spend Management Software for Receipts & Expenses | Expensify.” Accessed: Jul. 03, 2025. [Online]. Available: https://www.expensify.com/

[14] “AI Document Automation Software | ABBYY FlexiCapture,” ABBYY. Accessed: Jul. 03, 2025. [Online]. Available: https://www.abbyy.com/flexicapture/

[15] “Veryfi,” Veryfi. Accessed: Jul. 03, 2025. [Online]. Available: https://www.veryfi.com/

[16] “Online Bookkeeping Software for Small Businesses.” Accessed: Jul. 03, 2025. [Online]. Available: https://dext.com/en

[17] “#1 Receipt Scanner App & Free Mileage Tracker | Shoeboxed.” Accessed: Jul. 03, 2025. [Online]. Available: https://www.shoeboxed.com/

[18] “WellyBox | Organize Receipts and Invoices With The Power of AI,” WellyBox - The Best Receipt App for Small Businesses. Accessed: Jul. 03, 2025. [Online]. Available: https://www.wellybox.com/

[19] “home,” Gorilla Expense. Accessed: Jul. 03, 2025. [Online]. Available: https://www.gorillaexpense.com/