Master Thesis

Michał Zaręba
196218

Automated Extraction and Categorization of Product Information from Receipts

Diploma thesis in the field of
Information Science

Thesis under the supervision of
dr hab. inż. Leszek Chmielewski, prof. SGGW
Institute of Information Technology
Department of Artificial Intelligence

Warsaw, year 2017

WARSAW
UNIVERSITY
OF LIFE SCIENCES

Faculty of Applied
Informatics and
Mathematics

Declaration of the Thesis Supervisor

I hereby declare that this thesis has been prepared under my supervision, and I confirm that it meets the conditions for presenting this work in the procedure for the award of a professional title.

Date ..... Supervisor's signature .....

Declaration of the Author of the Thesis

Aware of legal liability, including criminal liability for submitting a false declaration, I hereby declare that this diploma thesis was written by myself and did not contain the content obtained in a manner inconsistent with applicable law, in particular the Act of 4 February 1994 on copyright and related rights (Journal of Laws of 2019, item 1231, as amended).

I declare that the submitted work has not previously been the basis for any procedure related to awarding a diploma or obtaining a professional title.

I certify that this work version is identical to the attached electronic version.

I acknowledge that the diploma thesis will be subject to the procedure of the anti-plagiarism.

Date ..... Author's signature .....

Streszczenie

Extraction and categorization

Słowa kluczowe – OCR, BERT, Transformer, NLP, thesis, implementation, SGGW, Warsaw University of Life Sciences

Summary

Ekstracja i kategoryzacja

lorem ipsum

Keywords – OCR, BERT, Transformer, NLP, thesis, implementation, SGGW, Warsaw University of Life Sciences

Contents

1 Introduction **11**

1.1 Motivation 11

1.2 Problem Statement 11

1.3 Objectives of the Study 12

1.4 Scope and Limitations 12

2 Literature Review **14**

2.1 Optical Character Recognition (OCR) Technologies 14

2.2 Semantic Text Embeddings for Receipt Item Categorisation 15

2.2.1 Count-based Methods 17

2.2.2 Prediction-based Methods 17

2.2.3 Sentence-BERT Embeddings 18

2.3 XGBoost Classifier 19

2.4 Challenges in OCR and NLP for Document Understanding 21

3 Methodology **23**

3.1 Introduction 23

3.2 Research Design 23

3.3 Data Sources, Collection and Preparation 24

3.3.1 Data Acquisition and Annotation 24

3.3.2 Data Pipeline and Loading 25

3.3.3 Class Distribution Analysis 25

3.3.4 Embedding Generation 27

3.4 Modeling and Analysis 27

3.5 Experimental Setup and Evaluation 28

3.6 Summary 29

4 Implementation **30**

4.1 Overview 30

4.2 Technology Stack 30

4.3 System Architecture 31

4.4 Component Implementation 32
4.4.1 DataLoader and Input Handling 32
4.4.2 Preprocessing Module 32
4.4.3 Embedding Interface 33
4.4.4 Training and Evaluation Pipeline 33
4.5 Training Configuration 34
4.6 Reproducibility and Setup 34
4.6.1 Project Structure 34
4.6.2 Installation Instructions 35
4.6.3 Configuration Management with Hydra 36
4.7 Summary 37

5 Evaluation and Results **38**
5.1 Experimental Setup 38
5.2 Evaluation Metrics 39
5.3 Result Analysis 39
5.3.1 Best Performing Model 39
5.3.2 Fine-Tuned Sentence-Transformer on Mixed Data 40
5.3.3 BERT-Based Model (Out-of-the-Box) 42

6 Discussion **43**
6.1 Interpretation of Results 43
6.1.1 Impact of fine-tuning. 44
6.1.2 Influence of GPT augmentation. 44
6.1.3 Comparison of Micro and macro metrics. 44
6.1.4 Practical implications. 44
6.2 Recommendations for Future Work. 44
6.3 Limitations of the Study 45

7 Conclusion **46**
7.1 Summary of Findings 46
7.2 Contributions to the Field 46

8 Appendices **48**
8.1 Example Receipt and Annotation 48
8.2 Implementation Examples 49

8.2.1 Text Preprocessing 49
8.2.2 OCR Pipeline 49
8.2.3 Sentence-Transformer Fine-tuning 50
8.2.4 Hydra-driven Model Training Pipeline 50
8.3 Additional Figures and Tables 51
8.3.1 XGBoost Hyperparameters by Model Type 51

Copyright notice

1 Introduction

1.1 Motivation

Tracking expenses and managing personal finances are important aspects of modern life. With an increasing number of daily transactions and a vast variety of products available, individuals face significant challenges in effectively monitoring their spending and managing their budgets. Although receipts contain valuable details that could help consumers analyse and control their expenses, the majority of shoppers either discard receipts shortly after purchase or find manual analysis too tedious and time-consuming. Automating the extraction and categorisation of product information from receipts could significantly simplify budget tracking and provide actionable insights into spending habits, enabling consumers to understand precisely where their money goes. Yet, few existing tools reliably process unstructured, Polish-language receipts at the line-item level, leaving a gap that this thesis aims to fill. A simple and efficient way to track expenses is therefore essential for individuals who wish to maintain a clear overview of their spending habits and make informed financial decisions.

1.2 Problem Statement

Most existing expense-tracking solutions focus on invoices, bank statements, or require manual input. Large corporations and organisations typically possess the budgets and technical resources to deploy robust, automated systems that extract and categorise expense data from structured documents such as invoices. For personal use, however, the most practical source of spending information remains paper receipts. Current receipt-based tools are often limited: many are designed exclusively for commercial purposes, lack support for languages other than English, or are inadequately trained to process Polish-language receipts accurately. Thus, a clear gap exists for a solution that automatically extracts and categorises line-item product details from receipts while accommodating the linguistic complexity of Polish.

1.3 Objectives of the Study

This study has **three concrete objectives**, each derived from the challenges set out in the problem statement:

1. Extract line-item text from Polish-language receipts using Tesseract OCR combined with lightweight heuristic pre- and post-processing.
2. Investigate two sentence-embedding models for representing product descriptions: the Polish BERT baseline, used without fine-tuning, and the `all-mini-l6-v2` model from the *sentence-transformers* library, fine-tuned on the domain dataset.
3. Train and evaluate an XGBoost classifier on the resulting embeddings, systematically comparing configurations to identify the combination that yields the highest macro-F1 score across predefined expense categories

Achieving these objectives requires careful integration of OCR and NLP components as well as robust hyper-parameter tuning to handle ambiguous, multi-word, or misspelled product names.

1.4 Scope and Limitations

The scope of this study is limited to developing a system that automatically extracts product information from receipts and categorises these products into predefined classes. The system specifically targets Polish-language receipts and will be evaluated chiefly on its ability to extract item prices accurately and assign correct product categories.

Limitations

- The system will *not* provide additional features such as longitudinal expense tracking, financial-report generation, or integration with external personal-finance applications.
- The scarcity of comprehensive, labelled Polish-language receipt datasets restricts the potential accuracy and generalisation capability of the developed models. Consequently, performance may degrade on receipt formats or lexical variants absent from the training data.
- OCR is applied without utilising spatial layout information (e.g. bounding boxes or reading order). Therefore, image preprocessing—cropping, alignment,

and noise reduction—is required to supply clean, line-level input to the OCR engine.

2 Literature Review

2.1 Optical Character Recognition (OCR) Technologies

Optical Character Recognition (OCR) is the process of converting textual information from scanned or photographed images into machine-readable formats. Traditional OCR techniques relied on template matching, statistical classification and structural analysis but offered limited adaptability to varying fonts and noisy inputs [6]. Modern systems employ deep-learning pipelines that combine convolutional neural networks (CNNs) for feature extraction with recurrent or attention-based architectures—such as LSTM or GRU layers—for sequence modelling [14].

Recent approaches add attention mechanisms so the network can focus dynamically on salient regions of an image. Although LayoutLM is primarily a document-understanding model rather than an OCR engine, it demonstrates how incorporating 2-D positional embeddings yields substantial gains on structured documents like receipts [17]. Despite these advances, attention-based models require large, annotated datasets; consequently the OCR tools evaluated in this study—Tesseract and PaddleOCR—operate at the text-line level and do not encode full document layout.

Tesseract. Tesseract is a widely adopted, open-source engine maintained by Google. It supports more than 100 languages, including Polish, and can be fine-tuned on domain-specific data for higher accuracy. Since version 4.0 it has featured an LSTM-based recogniser, improving robustness on noisy or multilingual inputs [15, 16]. Tesseract exposes numerous configuration options—segmentation modes, character whitelists and custom language packs—allowing practitioners to tailor the engine to narrow formats such as Polish retail receipts.

PaddleOCR. PaddleOCR is an open-source, deep-learning OCR framework developed by Baidu that currently supports more than eighty languages [4]. It offers an end-to-end, modular pipeline consisting of text detection, detected-box rectification, and text recognition, which simplifies component-level optimisation. Thanks to extensive data-augmentation strategies, the framework strikes a favourable balance between accuracy and inference speed, making it suitable for real-time and mobile scenarios.

Text detection is handled by *Differentiable Binarisation*, a lightweight segmentation network whose simplicity—augmented by minimal post-processing—maintains high recall and precision. The resulting bounding boxes are rectified via a text-direction classifier and then passed to the text-recognition module. This recogniser employs a convolutional recurrent neural network (CRNN) architecture, combining convolutional layers with recurrent layers to capture both local and sequential character features. Additional enhancements, such as a lighter backbone and task-specific augmentation, further improve performance while preserving low-latency requirements.

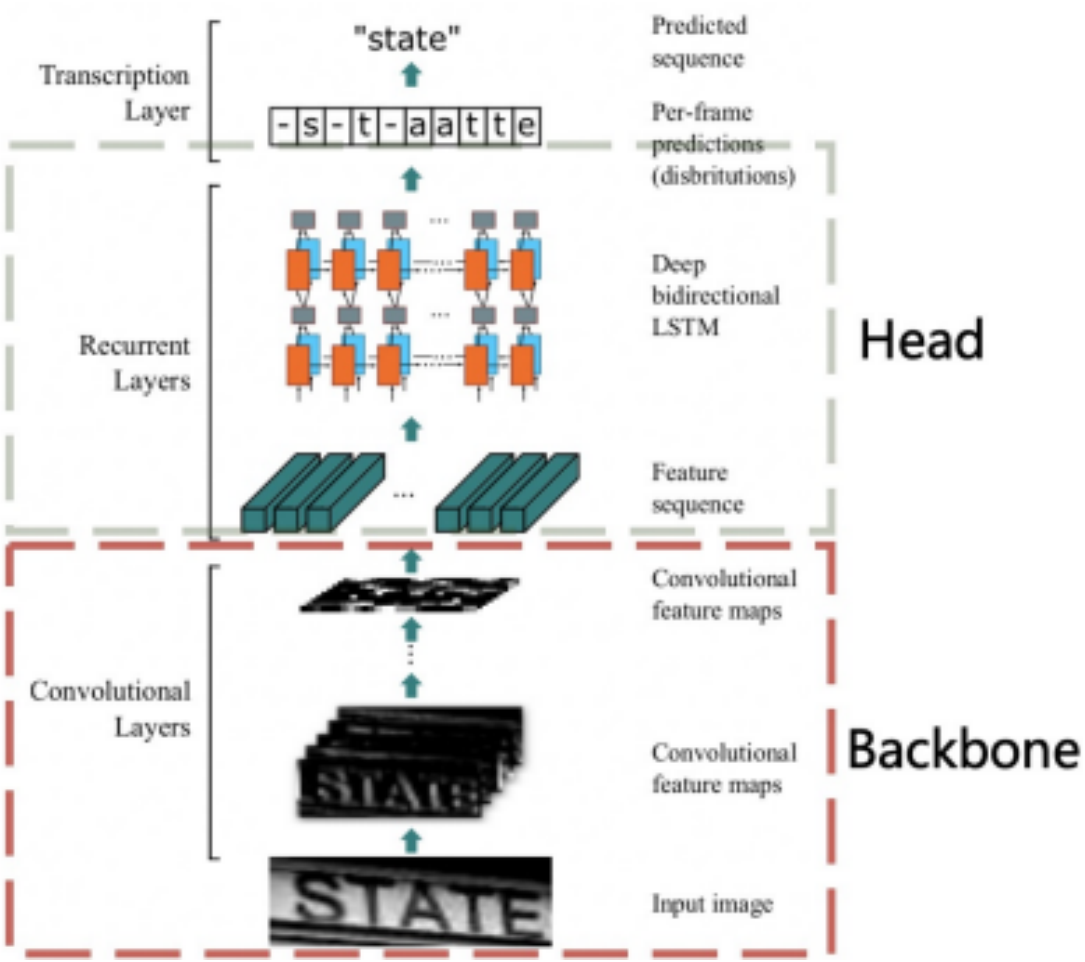

Figure 2.1. CRNN text recognition architecture [13].

2.2 Semantic Text Embeddings for Receipt Item Categorisation

To enable machine-learning models to analyse text, the text must first be converted into a numerical format—a process known as *vectorisation*. Word embeddings are

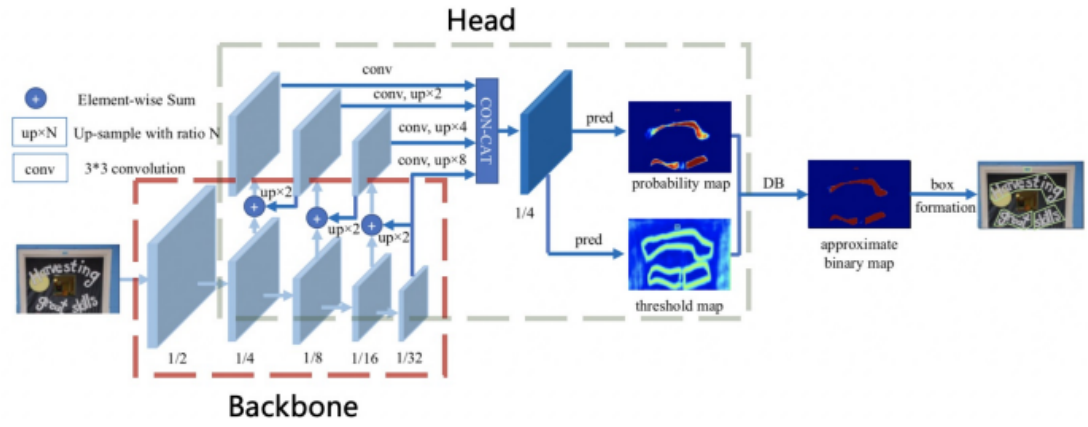

Figure 2.2. DBNet text detection architecture [8].

fixed-length vectors that encode semantic meaning and inter-word relationships; words with similar meanings occupy nearby positions in the embedding space [1].

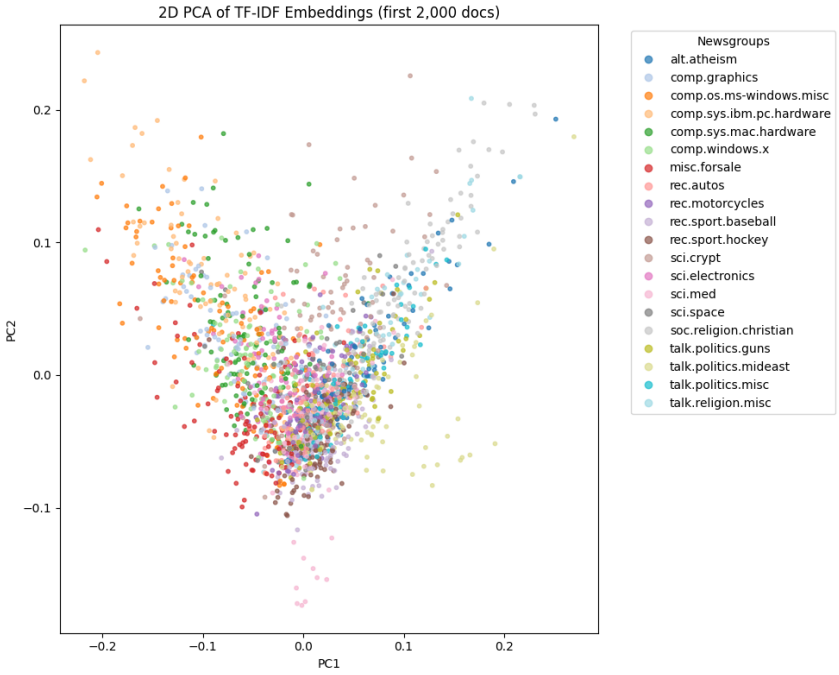

Figure 2.3. Two-dimensional projection of TF–IDF vectors for the *20 Newsgroups* dataset (scikit-learn).

There are several approaches to generating embeddings. Before embedding, the text is typically pre-processed: normalisation (e.g. lemmatisation or stemming) and tokenisation, which segments text into individual words or tokens [7].

Embedding methods are commonly divided into two families: *count-based* techniques (e.g. TF–IDF, GloVe) and *prediction-based* techniques (e.g. Word2Vec, fastText). The following subsections describe these groups and highlight their key differences.

2.2.1 Count-based Methods

Count-based methods rely on the premise that a word's meaning can be inferred from how often it co-occurs with other words in a given context. These methods typically construct a sparse matrix whose rows and columns enumerate the vocabulary, while each cell records the frequency of co-occurrence between word pairs. Two common variants are the *Bag-of-Words* (BoW) model and *Term Frequency–Inverse Document Frequency* (TF–IDF) [7].

Bag-of-Words. BoW represents a document as an unordered vector of raw word counts, ignoring word order and grammatical relations.

TF–IDF. TF–IDF assigns a weight to each word proportional to its frequency in the document but inversely proportional to its frequency across the whole corpus, thereby emphasising words that are especially informative for that document.

2.2.2 Prediction-based Methods

Prediction-based methods learn word representations by *predicting* a target word from its context—or vice versa—rather than by tallying raw co-occurrence counts. The resulting embeddings are stored in dense matrices, which are typically more parameter-efficient and capture semantic regularities better than the sparse matrices used by count-based techniques [7].

Two widely cited prediction-based models are *Word2Vec* and *BERT*. Word2Vec yields *static* embeddings: each word in the vocabulary is assigned a single, context-independent vector. By contrast, BERT—a transformer encoder—produces *contextual* embeddings whose values depend on the surrounding tokens.

Word2Vec. Word2Vec is trained under one of two architectures:

1. **Continuous Bag-of-Words (CBOW)** – predicts the centre (target) word from its surrounding context words.
2. **Skip-Gram** – predicts surrounding context words from a single target word.

Skip-Gram generally models rare words more faithfully, whereas CBOW offers greater computational efficiency and shorter training times [9].

BERT (Bidirectional Encoder Representations from Transformers) is a deep-learning model introduced by Google that generates *contextualised* word

embeddings by processing text simultaneously in both directions (left-to-right and right-to-left).

Unlike earlier models that produce static embeddings, BERT represents each word in the context of the entire sentence, thereby capturing polysemy and syntactic nuance.

The underlying transformer architecture employs self-attention to weight the contribution of each token relative to all others, allowing the model to *derive* complex semantic relationships. BERT is pre-trained in a two-part, self-supervised scheme:

1. **Masked Language Modelling (MLM)** – randomly masks a subset of input tokens and trains the network to predict the masked words from their context.
2. **Next-Sentence Prediction (NSP)** – learns to decide whether a given sentence *immediately follows* another, thereby improving the model's grasp of sentence coherence.

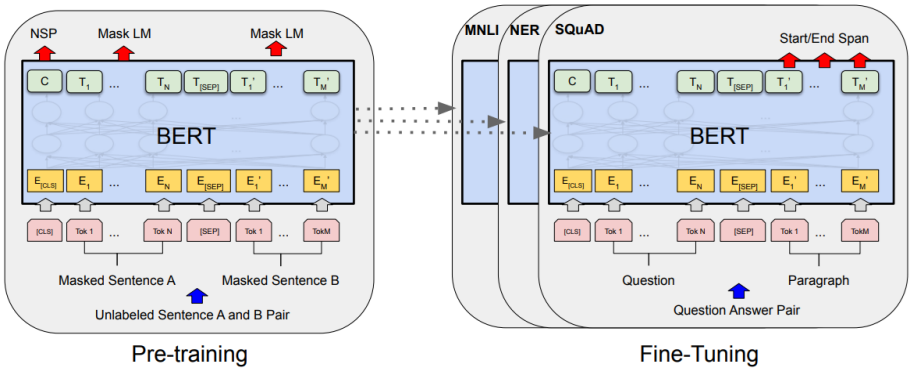

Figure 2.4. Pre-training objectives of BERT: MLM and NSP [3].

2.2.3 Sentence-BERT Embeddings

Sentence-BERT (SBERT) adapts the original BERT into a *Bi-BERT-encoder* architecture that maps sentences to dense vectors which can be compared with simple distance measures [11]. Two weight-sharing transformer towers encode each sentence independently; a subsequent pooling operation (typically mean pooling) yields a fixed-length embedding. Because the sentence vectors can be pre-computed, semantic search or clustering over n sentences requires only $O(n)$ encoder calls, in contrast to the $O(n^2)$ calls needed by cross-encoders that jointly process sentence pairs.

SBERT is trained with contrastive objectives such as *natural language inference* or *triplet loss*. During optimisation the cosine similarity between pairs of sentence embeddings is pushed high for semantically related sentences and low for unrelated

ones, thereby structuring the embedding space for downstream tasks such as receipt-item categorisation.

MiniLM backbone. MiniLM is a distilled transformer that compresses BERT into a lighter model by matching self-attention distributions and hidden-state relations while reducing depth and width [**wang2020minilm**]. Its efficiency makes it attractive for real-time or resource-constrained deployments without a dramatic drop in accuracy.

`all-mini-l6-v2`. The model `all-mini-l6-v2`—provided by the *sentence-transformers* library—is a six-layer SBERT distilled from `microsoft/MiniLM-L12-H384-uncased` and further fine-tuned on ~ 1 billion sentence pairs [12]. Fine-tuning employs a batch-wise contrastive objective: the cosine similarity is computed for each possible pair of sentences within a batch, and the model is optimised so that positive pairs obtain higher similarity scores than negative pairs. With 384-dimensional embeddings and a footprint of roughly 90 MB, `all-mini-l6-v2` offers an excellent speed–accuracy trade-off, which this thesis leverages for real-time categorisation of Polish receipt items.

Contextual embeddings produced by transformer-based models have shown strong performance in a wide range of NLP tasks, including text classification [3].

In the present study, embeddings generated by pre-trained BERT variants are used as feature vectors for a classifier that assigns receipt items to predefined expense categories.

2.3 XGBoost Classifier

XGBoost (*eXtreme Gradient Boosting*) is an open-source library that implements gradient- boosted decision trees. It consistently delivers state-of-the-art accuracy on a wide range of supervised-learning tasks [2].

Gradient boosting is an ensemble technique that, rather than fitting a single model, *builds* an initial weak learner and then iteratively fits additional learners that minimise the residual loss [10].

Key design features.

1. **Regularisation** – penalises overly complex trees, thereby reducing over-fitting and improving generalisation.

2. **Second-order optimisation** – uses both the gradient and the Hessian of the loss to approximate the objective, which accelerates convergence and improves estimation accuracy.
3. **Shrinkage (learning rate)** – scales each tree's contribution by a small factor after every boosting step, allowing the ensemble to learn gradually and robustly.
4. **Column subsampling** – randomly selects subsets of features when growing each tree, further mitigating over-fitting and speeding up training.

The final model is the sum of many shallow decision trees; each tree corrects the errors of the ensemble built so far, and their aggregated outputs yield the prediction.

Mathematically, the ensemble prediction for an instance x_i is

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i)$$

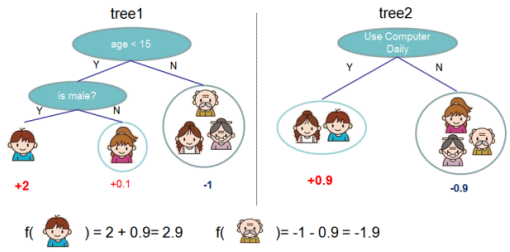

Figure 2.5. Tree-ensemble model [2].

During training, XGBoost optimises a second-order Taylor expansion of the loss, which determines the optimal structure and weights of the next tree. The per-iteration objective simplifies to

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T}\frac{\left(\sum_{i\in I_j} g_i\right)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma T$$

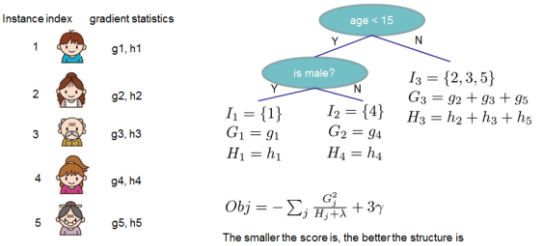

Figure 2.6. Structure-score calculation [2].

where, g_i and h_i denote the first and second-order gradient statistics of the loss with respect to the current prediction, while γ and λ are hyperparameters that penalise complex trees and thus curb over-fitting. In XGBoost these hyperparameters govern the trade-off between model complexity and generalisation.

Table 2.1. Principal XGBoost hyperparameters and their effects

Hyper-parameter	Role and effect
`learning_rate`	Scales each tree's contribution; lower values curb over-fitting but require more trees.
`max_depth`	Upper bound on tree depth; deeper trees capture complex patterns yet risk over-fitting.
`n_estimators`	Number of trees in the ensemble; more trees usually improve accuracy at the cost of runtime.
`subsample`	Fraction of training instances sampled per tree; values < 1 add stochasticity and reduce over-fitting.
`colsample_bytree`	Fraction of features sampled per tree; decorrelates trees similarly to random forests.
`min_child_weight`	Minimum sum of instance weights in a leaf; larger values force simpler splits.
`gamma`	Minimum loss reduction required to split; higher values prune trivial branches.
`reg_alpha`	L_1 penalty on leaf weights; promotes sparsity by driving some weights to zero.
`reg_lambda`	L_2 penalty on leaf weights; discourages extreme weights and thus over-fitting.

Tuning these settings balances close fit to the training data against the need for a simpler, more generalisable model

2.4 Challenges in OCR and NLP for Document Understanding

The integration of OCR and NLP technologies for document understanding presents several challenges, particularly with complex documents such as retail receipts. These challenges include variability in receipt formats, image noise and distortion, and language-specific characteristics. Even state-of-the-art OCR systems struggle to maintain high accuracy on low-quality images but basic image preprocessing like binarisation, denoising, and skew correction, can improve input quality and thus OCR performance.

To enhance downstream understanding, some models employ *2-D positional embeddings*. These embeddings encode each token's horizontal (x) and vertical (y) coordinates whereas standard (1-D) positional embeddings capture only sequential order. Incorporating spatial layout information helps the network model document structure more accurately, which benefits tasks such as key-information extraction [18].

A complementary strategy is *text restoration*. For example, an Action-Prediction

Model predicts character-level correction actions on the raw OCR output; applying these corrections raised NER performance from 0.59 to 0.895 F1 granting a 76 % relative gain in one study [5]. Such restoration can likewise boost text-classification accuracy.

A critical bottleneck for OCR research is the scarcity of *labelled* datasets: every document image must be paired with precise text annotations, an expensive requirement for complex receipts [18]. Limited data heightens the risk of over-fitting, where models perform well on the training set yet generalise poorly to unseen inputs. Synthetic data generation offers a promising remedy. Tools such as *Genalog* and recent LLM-based generators can create large, diverse receipt corpora, alleviating the annotation burden and improving model robustness [5].

3 Methodology

3.1 Introduction

This chapter outlines the methodology used to develop and evaluate a system that automatically extracts product information from *Polish*-language receipts and categorises those products into expense-related classes.

The workflow comprises five stages: data acquisition, preprocessing, embedding generation, classification, and experimental evaluation.

The project is structured to support a systematic comparison of alternative embedding strategies and to quantify their impact on classification performance, while remaining scalable and adaptable to future enhancements.

3.2 Research Design

The primary objective of this study is to develop and rigorously evaluate an end-to-end system for extracting and categorizing product information from receipts. To achieve this, the research design adopts a modular approach: each component (OCR, embedding generation, and classification) can be developed and tested independently, while still integrating into a unified workflow. This structure not only enables targeted evaluation of individual modules but also supports direct comparison across different models and techniques. Finally, model performance is assessed under two experimental which is 8-class categorization tested with both original and GPT-augmented training data—to measure robustness across varying levels of complexity and data composition, with controlled variation of input features and training data composition.

The dataset used in this study is relatively small as it consists of only 26 scanned receipts, yielding a total of 292 individual line-item entries. I applied a standard 70/30 train/test split. Then the training is split in 90/10 ratio for a validation script, resulting in 184 entries for training, 88 entries for testing as a validation set used for finetuning. Polish-specific textual challenges further complicate the task. First, diacritical marks (e.g., “ł”, “ó”, “ą”, “ę”) are frequent and, if misrecognized by

OCR, can lead to incorrect tokenization and erroneous embeddings. Second, receipts often employ numerous abbreviations and shortcuts—such as “kg” for kilograms or truncated product names (ZIEL, KAJZ etc.)— introducing high lexical variability. These factors necessitate targeted preprocessing (e.g., regex rules for diacritic restoration and expansion of common abbreviations) and robust embedding strategies to mitigate recognition errors and vocabulary fragmentation.

Model performance is evaluated using the macro-averaged F1 score, which balances precision and recall across all classes. As a baseline, I implement a straightforward pipeline comprising TF-IDF vectorization for feature representation, and a logistic regression classifier. The OCR component is the same in both cases as it is a complicated task to implement a new one, that would be able toprovide data that could be analyzed by the model. This baseline enables quantification of the performance gains achieved through advanced embedding models and XGBoost classification.

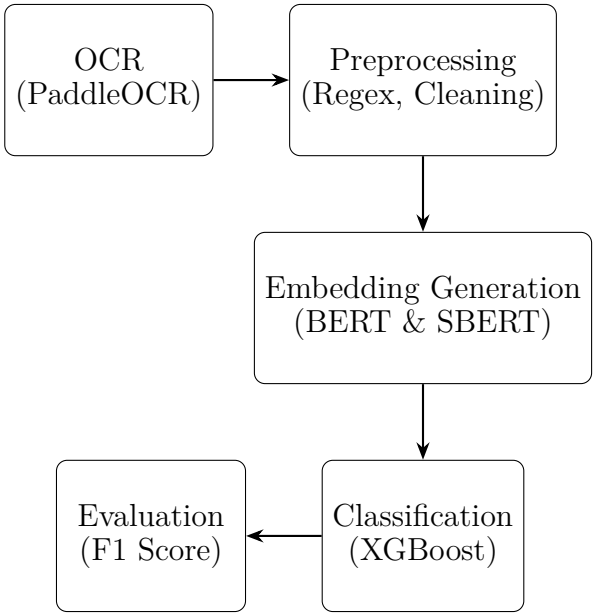

Figure 3.1. High-level flowchart of the receipt processing and classification pipeline

3.3 Data Sources, Collection and Preparation

3.3.1 Data Acquisition and Annotation

The primary dataset consists of 292 line-item entries extracted from 26 scanned receipts using PaddleOCR. To address class imbalance, the dataset was enriched

with GPT-generated synthetic examples, achieving a more uniform distribution across categories. Each entry was manually annotated with a product category—according to a 8-class schema (Food, Beverages, Snacks, Hygiene, Housing, Healthcare, Discount, Other)—and the corresponding cost. All annotations were consolidated into a single CSV file for subsequent processing.

3.3.2 Data Pipeline and Loading

Our pipeline begins by reading the annotated CSV, which contains columns:

- **Text**: Extracted text from the receipt line item.
- **Category**: Annotated category label (8 classes).
- **Cost**: Cost associated with the product.

A label encoder converts the categorical labels into numeric codes. To ensure consistency and reduce noise in the textual input, a set of normalization and cleaning steps was applied to each text line. The objective was to eliminate irrelevant artifacts and standardize the structure of the text prior to embedding generation and classification.

Punctuation characters were replaced with spaces to clearly separate word tokens and improve downstream tokenization. Consecutive whitespace characters were collapsed into a single space to maintain uniform spacing. Citation-like patterns were removed to eliminate non-linguistic references commonly produced by OCR. All non-alphanumeric characters—excluding whitespace—were stripped to retain only meaningful text content. Numerical digits were removed to reduce the influence of context-irrelevant pricing or codes, and single-character words were excluded, as they typically carry little semantic value in this domain.

These preprocessing operations helped standardize the textual data and ensure that only informative content was passed to the embedding models and classifier. Finally, the balanced dataset is split into training (70 %) and test (30 %) sets, ensuring that each subset maintains the overall class distribution.

3.3.3 Class Distribution Analysis

Figure 3.2 presents a comparison of two distinct datasets used in this study: the original OCR-extracted and manually annotated receipt data, and a supplementary dataset generated using GPT. The latter was constructed to ensure equal representation

across all defined classes and is therefore referred to as the "balanced" GPT dataset.

In the case of the 8-class taxonomy, the original receipt data show a strong class imbalance. The *Food* category dominates, comprising more than half of all annotated items, while other categories, such as *Hygiene* or *Healthcare*, are represented by only a few examples. This skew is expected, as food is naturally the most frequent part of daily purchases. However, such an imbalance poses a challenge for classification, as models trained on these data tend to favor majority classes. In contrast, the GPT-generated dataset contains approximately the same number of samples per class. While this is useful for introducing diversity and covering underrepresented categories, the uniformity of the GPT dataset does not reflect the real-world class distribution present in actual receipts.

Although balancing categories is useful during data augmentation, the use of such synthetic data for model training requires alignment with the distribution observed in real receipts. If the class distribution between the original and GPT datasets differs significantly, the resulting model may overfit to artificial class proportions rather than learning meaningful class distinctions. Therefore, prior to training, the GPT-generated data should be resampled to reflect the empirical class distribution of the original data.

Maintaining comparable class distributions across datasets ensures that the classifier is exposed to realistic category proportions and that performance comparisons are based on model effectiveness rather than artifacts of imbalanced or overly uniform training inputs.

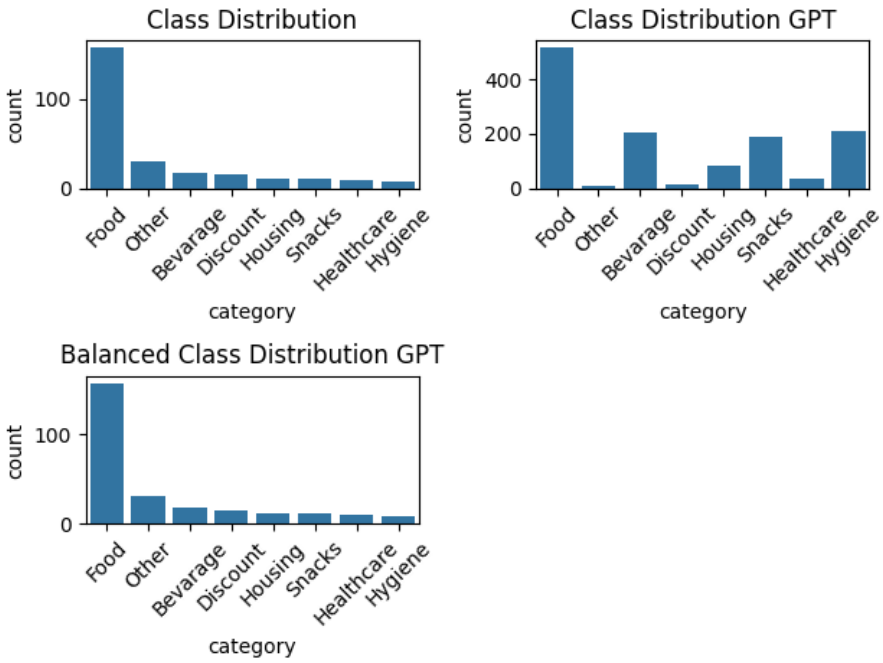

Figure 3.2. Original, GPT-augmented, and balanced (bottom) class distributions.

3.3.4 Embedding Generation

Two distinct embedding approaches are evaluated and compared in this study. The first utilizes the pretrained Polish BERT model `dkleczek/bert-base-polish-cased-v1`, which generates token-level contextual embeddings based on a broad Polish-language corpus. This model has not been fine-tuned for the receipt classification task and serves as a strong general-purpose baseline.

The second approach uses the `all-MiniLM-L6-v2` Sentence-BERT model, which is a lighter, sentence-level embedding model. This version has been further fine-tuned specifically on receipt-related data to better capture domain-specific semantics relevant to product categorization.

These two models are not used together in a single pipeline. Instead, they are compared in parallel to assess how domain adaptation and model size influence classification performance. In both cases, the resulting embeddings are used as input features for the XGBoost classifier.

3.4 Modeling and Analysis

This section describes the structure of the modeling experiments and the rationale behind the comparison strategy. The classification task is handled using the XGBoost

algorithm, which operates on fixed-length vector embeddings generated from individual receipt line items. The model receives as input either general-purpose or fine-tuned sentence embeddings, depending on the experiment.

Two distinct embedding sources are evaluated independently. The first uses the Polish BERT model `dkleczek/bert-base-polish-cased-v1`, which is pretrained on a general corpus and not adapted to the receipt classification task. The second approach uses the `all-MiniLM-L6-v2` Sentence-BERT model, which has been further fine-tuned on a domain-specific receipt dataset. These models are not used jointly, but rather compared in parallel to assess their effectiveness in representing receipt-level semantics.

To ensure that each model configuration performs at its best, the XGBoost hyperparameters are not fixed across experiments. Instead, the classifier is tuned separately for each embedding approach. Hyperparameters such as learning rate, maximum tree depth, number of estimators, and regularization strength are optimized individually using cross-validation on the training set. This allows each embedding–classifier combination to be evaluated under its most favorable settings, ensuring that the comparison reflects true model capability rather than differences in optimization constraints.

Classification is performed in 8-class taxonomy for both of the embedding types. In each setup, training and testing splits are kept consistent across experiments to allow fair comparison. Model performance is measured using the macro-averaged F1 score, which gives equal weight to each class, regardless of frequency. This is especially important given the underlying imbalance in real-world receipt data.

Overall, the modeling strategy is designed to evaluate how both the choice of embeddings and the label schema affect classification accuracy, while giving each setup the opportunity to achieve optimal performance through tailored hyperparameter tuning.

3.5 Experimental Setup and Evaluation

All experiments were managed using the Hydra framework, which provided structured configuration management and ensured reproducibility across runs. The training data included both the original receipt dataset and GPT-generated samples, with the latter resampled to match the class distribution of the original data.

Four classification scenarios were evaluated, corresponding to two embedding

strategies—Polish BERT and fine-tuned Sentence-BERT—each tested on both the original and GPT-augmented training sets. This experimental setup enables a direct comparison of how embedding quality and data composition affect final model performance.

Evaluation metrics, including accuracy and macro-averaged F1 scores, are reported in Chapter V.

3.6 Summary

The methodology presented in this chapter combines OCR-based text extraction, manual data annotation, targeted text preprocessing, and embedding-based feature generation with an XGBoost classifier. The experimental framework is structured to support a systematic comparison between two embedding strategies and to evaluate classification performance under different training conditions. This design allows for reliable assessment of how embedding quality and data composition influence the effectiveness of multi-class product categorization from receipt data.

4 Implementation

4.1 Overview

The implemented system is designed as a modular pipeline, with clearly defined components for data loading, preprocessing, embedding generation, model training, and evaluation. Each component is developed independently and integrated through centralized configuration management provided by Hydra.

This modular structure allows easy substitution and evaluation of different embedding methods, classification algorithms, and data sources without requiring extensive code changes. All configurations, including hyperparameters, dataset versions, and experimental setups, are centrally managed by Hydra, ensuring consistent and reproducible experimentation.

Implemented entirely in Python, the system emphasizes code readability, extensibility, and reproducibility. It supports multiple embedding models, seamlessly switching between pretrained and fine-tuned variants. Experiment results, logs, and artifacts are systematically organized, simplifying comparison and analysis.

4.2 Technology Stack

The implementation leverages a carefully selected set of technologies and libraries to ensure robust, efficient, and scalable functionality. Below is an overview of the core components used in the system:

- **Programming Language:** Python 3.10 was chosen due to its extensive ecosystem and strong support for machine learning, NLP, and computer vision tasks.
- **OCR Framework:** PaddleOCR was selected for its modular architecture, robust multilingual capabilities, and strong support for Polish text recognition.
- **Embedding Models:**
  - `dkleczek/bert-base-polish-cased-v1` for generating general-purpose contextual embeddings in Polish.

  - `all-MiniLM-L6-v2` Sentence-BERT model, specifically fine-tuned on receipt-related textual data.
- **Classification Framework:** XGBoost was chosen for its efficiency, scalability, and consistently high performance in supervised classification tasks.
- **Data Processing and ML Libraries:**
  - `Transformers`: Handling pretrained language models.
  - `SentenceTransformers`: Fine-tuning and generating semantic embeddings.
  - `scikit-learn`: Data preprocessing, evaluation metrics, and utility functions.
- **Configuration Management:** Hydra framework for centralized and reproducible experiment configuration management.
- **Visualization Libraries:** Matplotlib and Seaborn for data visualization, exploratory analysis, and evaluating model performance.
- **System Dependencies:**
  - `CUDA`: GPU acceleration during training and inference.
  - `PyTorch`: Deep learning backend framework.
  - `OpenCV`: Image preprocessing tasks, including resizing, binarization, and noise reduction.

This technology stack was selected to balance performance, ease of integration, and extensibility, enabling reliable experimentation and reproducible outcomes.

4.3 System Architecture

The implemented system employs a modular architecture, structured into distinct components to ensure clarity, flexibility, and scalability. Each module is responsible for a specific task in the processing pipeline, and data flows sequentially through these interconnected modules:

- **OCR Module:** Extracts textual information from scanned receipt images using PaddleOCR.
- **Preprocessing Module:** Cleans, normalizes, and prepares the extracted text by addressing language-specific issues such as Polish diacritics, abbreviations, and common OCR-induced errors.
- **Embedding Module:** Converts the preprocessed textual data into numerical embeddings, utilizing pretrained language models like Polish BERT or fine-

tuned Sentence-BERT.

- **Classification Module:** Uses XGBoost to categorize products based on generated embeddings into predefined expense-related classes.
- **Evaluation Module:** Measures and assesses the classifier's performance using metrics such as accuracy and macro-averaged F1 scores.

This modular design supports independent development, evaluation, and substitution of individual components, enabling straightforward integration of new models or processing techniques. The detailed data flow and interactions between these modules are illustrated in the "Research Design" section (Figure 3.1).

4.4 Component Implementation

4.4.1 DataLoader and Input Handling

The DataLoader module loads and prepares input data for further processing. It accepts CSV files containing receipt line-item texts, annotated product categories, and associated costs. The module supports two types of input data: original OCR-extracted receipts and GPT-generated synthetic receipts, ensuring unified preprocessing for both.

To mitigate class imbalance within the training dataset, the DataLoader implements undersampling by randomly taking samples from different categories until they share the same distribution. This approach balances the class distribution, improving classifier performance and reducing bias towards dominant categories.

4.4.2 Preprocessing Module

The preprocessing module is responsible for cleaning and standardizing the raw textual data obtained from the OCR output. It utilizes regular expressions to remove unwanted elements such as punctuation marks, numerical artifacts, and citation-like patterns. The text is then normalized by converting all characters to lowercase and removing redundant whitespace characters.

Additionally, specific preprocessing steps tailored to the Polish language are implemented. These include the restoration of Polish diacritics and the expansion of common abbreviations frequently encountered in receipts. These transformations ensure consistent formatting of text, facilitating effective embedding generation and

subsequent classification tasks.

4.4.3 Embedding Interface

The Embedding Interface module provides a standardized mechanism for transforming preprocessed textual data into numerical vector representations. It integrates two embedding strategies, both leveraging pretrained transformer-based models:

- **Polish BERT (`dkleczek/bert-base-polish-cased-v1`):** Generates token-level contextual embeddings suitable for capturing general semantic and syntactic information.
- **Sentence-BERT (`all-MiniLM-L6-v2`):** Produces sentence-level semantic embeddings, specifically fine-tuned on receipt-related data to capture domain-specific nuances.

Internally, the module manages:

- Model initialization and loading.
- Tokenization using model-specific tokenizers.
- Batch processing for computational efficiency.
- Extraction and aggregation of embedding vectors.

This standardized processing ensures consistency and compatibility for downstream tasks, such as classification.

4.4.4 Training and Evaluation Pipeline

The Training and Evaluation pipeline implements supervised categorization using the XGBoost gradient boosting algorithm. Its responsibilities include:

- Training the classifier using numerical embeddings as input features and annotated product categories as target labels.
- Splitting data into distinct training and validation subsets to facilitate effective hyperparameter optimization via cross-validation.
- Centralized experiment management through Hydra configuration files, specifying hyperparameters, dataset paths, and model parameters to ensure reproducibility and consistency across runs.

The evaluation component computes critical performance metrics, specifically accuracy and macro-averaged F1 scores, allowing systematic and objective comparison across various embedding approaches and experimental configurations.

4.5 Training Configuration

The training configuration for the experiments is as follows:

- **Hardware:**
  - CPU: AMD Ryzen 7600
  - GPU: NVIDIA RTX Titan
  - RAM: 32 GB
- **Software:**
  - Operating System: Ubuntu 22.04
  - Python Version: 3.10
  - CUDA Version: 11.8
  - PyTorch Version: 2.0
- **Reproducibility:**
  - Random Seed: 42 Fixed for all experiments to ensure reproducibility.
  - Logging: All experiment logs, metrics, and configurations are stored using Hydra's output directory structure.
- **Approximate Training Time:** The XGBoost classifier training is very fast and typically completes within seconds. The longest process is embedding fine-tuning, which takes approximately 10 to 30 minutes depending on the dataset size and model complexity.

4.6 Reproducibility and Setup

To ensure reproducibility and ease of setup, the project follows a structured organization and provides clear installation instructions.

4.6.1 Project Structure

The project is organized as follows:

- **src/**: Main Python package
  - **analysis/**: Jupyter notebooks for exploratory data analysis, visualization, and prototyping (e.g. `data_analysis.ipynb`, `thesis_visualization.ipynb`, `tokenization.ipynb`).
  - **conf/**: Hydra configuration files and parameter definitions

        * `config.yaml`
        * `params/` (additional YAML fragments)
    - **data/**: Code-organized data and annotations
        * `annotations/`
        * `images/`
    - **model/**: Model code and artifacts
        * `dataloader.py`, `classifier.py`, `embeddings.py`, `ocr.py`
        * `xgboost_model.json`
    - **scripts/**: Stand-alone Python scripts (e.g. `annotation_pipeline.py`, etc.)
    - **utils/**: Utility functions and helpers (`utils.py`)
- **main.py**, **train_model.py**: Top-level entry points for running the pipeline and training the model.
- **pyproject.toml**, **uv.lock**: Project metadata and locked dependencies for UV.
- **README.md**, **.gitignore**: Project documentation and Git ignore rules.

4.6.2 Installation Instructions

Follow these steps to set up the environment and reproduce the experiments:

1. Install UV (Ultrafast Virtualenv) if not already installed:

   ```
   curl -LsSf https://astral.sh/uv/install.sh | sh
   ```

2. Install dependencies for an existing project:

   ```
   cd /path/to/project
   uv sync
   ```

   This reads your committed `uv.lock` (or `pyproject.toml`) and creates/populates `.venv/` automatically.

3. Ensure CUDA is installed for GPU acceleration:

   ```
   nvcc --version
   ```

   Verify that the installed CUDA version matches the PyTorch version.

4. Run experiments using Hydra configuration files:

   ```
   uv run python scripts/train.py -- --config-name=config.yaml
   ```

   (`uv run` will auto-sync your environment before executing.)

These steps and details ensure that the experiments can be reproduced reliably on similar hardware and software configurations.

4.6.3 Configuration Management with Hydra

To ensure reproducibility and maintain a clean separation between code and experiment settings, all configurations were managed using the Hydra framework. This allowed for flexible modification of parameters such as model type, data sources, embedding strategies, and training hyperparameters, without changing the core implementation.

Each experiment was launched with a dedicated YAML configuration file. An example configuration is shown below:

Listing 4.1. Sample Hydra configuration

```
params:
  learning_rate: 0.1
  max_depth: 6
  n_estimators: 100
  subsample: 0.8
  colsample_bytree: 0.8
  min_child_weight: 5
  gamma: 0.1
  reg_alpha: 0.1
  reg_lambda: 0.1

embeddingmodel: 'bert'
finetune: False
gptdatapath: '/path/to/gpt_generated_data8classes.csv'
datapath: '/path/to/annotations_8classes.csv'

defaults:
  - params: params_best_8classes
  - override hydra/sweeper: optuna
```

Hydra automatically stores every run in a separate output directory, including:

- a full copy of the configuration used,
- all generated logs,
- result metrics.

This structure guarantees reproducibility and simplifies tracking the performance of different model configurations over multiple experiments.

4.7 Summary

The implemented system adopts a modular architecture consisting of independent components for OCR, preprocessing, embedding generation, classification, and evaluation. This modularity enables flexible experimentation and easy integration of alternative models or techniques. Configuration management with Hydra ensures reproducibility, simplifies hyperparameter tuning, and supports structured comparisons across different experimental setups.

5 Evaluation and Results

5.1 Experimental Setup

To optimize model performance, a hyperparameter search was conducted using the **Optuna** framework with **1000 trials**. The goal was to maximize the overall F1 Score on the test set. The search space covered key parameters of the XGBoost classifier and included additional categorical choices related to embeddings, data and model configuration. Sampling was performed using TPE Sampler with seed set to 42. The parameter values considered were:

- **learning_rate**: {0.01, 0.05, 0.1, 0.2, 0.3}
- **max_depth**: {3, 5, 7, 10, 15, 20}
- **n_estimators**: {50, 100, 200, 300, 500, 750, 1000}
- **subsample**: {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
- **colsample_bytree**: {0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
- **min_child_weight**: {1, 2, 3, 5, 7, 10}
- **gamma**: {0.0, 0.1, 0.5, 1.0, 2.0, 5.0}
- **reg_alpha**: {0.0, 0.1, 0.5, 1.0, 2.0, 5.0}
- **reg_lambda**: {0.0, 0.1, 0.5, 1.0, 2.0, 5.0}
- **embeddingmodel.model_path**: {mini_finetuned_base, mini_finetuned_augmented, bert}
- **augument_classifier**: {True, False}
- **balance_data**: {True, False}

This setup enabled the optimization process to explore a broad range of configurations in a structured and efficient way. Although the sampler could occasionally converge to local optima, overall model performance improved substantially through hyperparameter fine-tuning and selection of the best-performing configuration.

5.2 Evaluation Metrics

To evaluate model performance, standard metrics are used: accuracy, precision, recall, and F1 score. Accuracy reflects the overall proportion of correct predictions but can be misleading in imbalanced datasets. Precision and recall offer a more detailed view—precision measures how many predicted positives are correct, while recall indicates how many actual positives are captured.

The main evaluation metric in this study is the micro-averaged F1 score, also referred to as the overall F1 score. It aggregates true positives, false positives, and false negatives across all classes and is particularly appropriate when class imbalance exists, as it reflects the model's global effectiveness.

In addition to this, macro-averaged F1 score is reported. It calculates F1 scores per class and averages them, giving equal weight to all categories. This provides insight into the model's balance across both frequent and rare classes.

Reporting both micro and macro F1 scores allows for fair evaluation: the former captures overall performance, while the latter highlights how well the model handles less-represented categories.

5.3 Result Analysis

5.3.1 Best Performing Model

The best results were achieved using the `all-MiniLM-L6-v2` model from the Sentence-Transformers library. This embedding model was fine-tuned exclusively on real receipt data, without using any GPT-generated examples during the embedding stage. The final classification was performed using an XGBoost model trained on a combination of real and GPT-augmented data. The GPT samples were balanced across classes to improve generalization on underrepresented categories.

This configuration achieved an overall accuracy of 79.55%, a micro-averaged (overall) F1 score of 0.8053, and a weighted F1 score of 0.81. The macro-averaged F1 score reached 0.70, confirming strong and consistent performance across both frequent and rare classes.

Model Configuration

The XGBoost classifier was tuned using the following hyperparameters:

- `learning_rate`: 0.01 `max_depth`: 5 `n_estimators`: 1000
- `subsample`: 1.0 `colsample_bytree`: 0.7
- `min_child_weight`: 2 `gamma`: 2.0
- `reg_alpha`: 0.0 `reg_lambda`: 5.0

Overall Performance

The performance summary for this configuration is presented in Table 5.1. The model demonstrates strong micro and macro F1 scores, making it the most balanced configuration tested. It also confirms that fine-tuning the embedding model on domain-specific real data is more effective than relying on pretrained models alone.

Table 5.1. Overall performance – Best model (`all-MiniLM-L6-v2` fine-tuned on real data, XGBoost trained on real + GPT-balanced data)

Metric	Macro Average	Weighted Average	Overall Accuracy
Precision	0.74	0.83	0.7955
Recall	0.70	0.80	0.7955
F1 Score (Macro)	0.70	0.81	0.7955
F1 Score (Micro)		0.8053	
Overall Accuracy		0.7955	

Class-Level Performance

The detailed classification report is shown in Table 5.2. High F1 scores are achieved for the dominant classes *Food* (0.88), *Discount* (0.91), and *Beverage* (0.83). These categories benefit from strong representation in the dataset (Food) or consistent linguistic patterns (Discount,Bevarage). More challenging classes like *Housing* and *Healthcare* achieved moderate performance (F1 = 0.50 and 0.47 respectively).

In summary, this configuration provides the most balanced and robust performance. Fine-tuning the embedding model exclusively on real data appears to preserve domain-specific semantics, while the use of GPT-balanced data for classifier training helps correct for class imbalance. This model sets a reliable benchmark for further improvements in receipt-based product categorization.

5.3.2 Fine-Tuned Sentence-Transformer on Mixed Data

This configuration used the `all-MiniLM-L6-v2` model fine-tuned on a combination of real and GPT-augmented data. The XGBoost classifier, however, was trained

Table 5.2. Full classification report – Best model

Class	Precision	Recall	F1 Score	Support
Beverages (0)	1.00	0.71	0.83	7
Discount (1)	1.00	0.83	0.91	6
Food (2)	0.89	0.87	0.88	55
Healthcare (3)	0.36	0.67	0.47	6
Housing (4)	0.50	0.50	0.50	4
Other (5)	0.67	0.60	0.63	10
Macro Avg	0.74	0.70	0.70	88
Weighted Avg	0.83	0.80	0.81	88
Overall Accuracy		0.7955		
Micro Avg F1 Score		0.8053		

only on real receipt data. This setup achieved an accuracy of 78.41%, a weighted F1 score of 0.80, and a macro-averaged F1 score of 0.77. Although the overall (micro) F1 score was slightly lower (0.7967) than the best-performing model, the macro-average indicates better class-wise consistency mostly due to high improvemend in Healthcare class

Table 5.3. Full classification report – Sentence-Transformer fine-tuned on real + GPT data, XGBoost trained on real data

Class	Precision	Recall	F1 Score	Support
Beverages (0)	0.88	1.00	0.93	7
Discount (1)	1.00	0.83	0.91	6
Food (2)	0.88	0.78	0.83	55
Healthcare (3)	1.00	0.83	0.91	6
Housing (4)	0.50	0.50	0.50	4
Other (5)	0.41	0.70	0.52	10
Macro Avg	0.78	0.77	0.77	88
Weighted Avg	0.82	0.78	0.80	88
Overall Accuracy		0.7841		
Micro Avg F1 Score		0.7967		

This model shows better macro-averaged F1 performance compared to others, reflecting more balanced results across all classes, including underrepresented ones. The problem with this result is that the embedding model showed opposite behavior compared to the best model, thus we could suspect that the model is overfitting to the GPT-generated data, which is not representative of real-world receipts.

5.3.3 BERT-Based Model (Out-of-the-Box)

This experiment evaluates the performance of the pretrained Polish BERT model (`dkleczek/bert-base-polish-cased-v1`) used without task-specific fine-tuning. Despite its significantly larger architecture compared to `all-MiniLM-L6-v2`, the model achieves lower performance across key metrics and slower inference speed, making it a less practical choice in this context.

The model reached an accuracy of 72.73% and a weighted F1 score of 0.7073. While these values are comparable to other configurations, the macro-averaged F1 score dropped to 0.51—substantially lower than previous model. This decline highlights its inability to generalize well across all classes, especially those with lower support or more semantic variability.

As expected, the *Food* category continues to dominate in terms of recall (0.89) and F1 score (0.83). However, several other classes exhibit weaker performance. For instance, the *Beverage* class yields an F1 score of only 0.18, while the *Housing* class is not predicted correctly at all. These issues result in poor macro-averaged metrics, confirming that the model disproportionately favors majority classes and struggles with semantically diverse or underrepresented labels.

It is also important to emphasize the inefficiency of this model. BERT's size significantly increases inference time without offering a performance advantage. In contrast, the smaller `all-MiniLM-L6-v2`, fine-tuned on receipt data, achieves superior scores while delivering much faster predictions—making it a more suitable choice for real-world applications.

A full classification breakdown for this BERT configuration is provided in Appendix 8.1.

6 Discussion

6.1 Interpretation of Results

Table 6.1 summarizes and compares the three evaluated configurations. The best-performing setup (0.81 micro-F1) utilizes the `all-MiniLM-L6-v2` embedding model fine-tuned solely on real receipt data, paired with an XGBoost classifier trained on combined real and GPT-augmented datasets. The mixed-data embedding fine-tuning achieves a higher macro-F1 score (0.77), reflecting more balanced classification performance across categories, particularly improving classes such as *Healthcare*. The out-of-the-box Polish BERT model performs poorly across all metrics, also incurring a substantial inference time penalty.

Table 6.1. Performance comparison of the evaluated model pipelines

Pipeline	**Accuracy**	**F1$_{\text{micro}}$**	**F1$_{\text{macro}}$**	**Main findings**
MiniLM (real) + XGBoost (real+GPT)	0.80	0.81	0.70	Best overall accuracy; GPT augmentation improves minority-class recall.
MiniLM (real+GPT) + XGBoost (real)	0.78	0.80	0.77	Highest macro-F1; balanced across classes, notably enhanced *Healthcare*.
BERT (pre-trained) + XGBoost (real)	0.73	0.71	0.51	Slow inference; weak generalization, favors majority class (*Food*).

6.1.1 Impact of fine-tuning.

Domain-specific fine-tuning proves more effective than using larger but unadapted models. Both MiniLM variants significantly outperform the larger BERT model used without fine-tuning.

6.1.2 Influence of GPT augmentation.

Differences between augmentation strategies suggest distinct effects:

- Fine-tuning embeddings on combined real and GPT data likely shifts the embedding representations toward the GPT-generated distribution. Thus, introducing more GPT samples at the classifier training stage could adversely affect performance on real receipt examples due to mismatched data distributions.
- Fine-tuning embeddings exclusively on real receipt data maintains a stable representation space for genuine receipt semantics. In this scenario, adding balanced GPT data at the classifier stage enhances generalization without negatively affecting the embedding manifold, thus benefiting overall model performance.

6.1.3 Comparison of Micro and macro metrics.

The optimal configuration demonstrates the highest micro-F1, emphasizing accurate predictions aligned with user-visible performance. The mixed-data fine-tuning setup leads in macro-F1, showcasing improved classification balance across diverse categories.

6.1.4 Practical implications.

Compact, domain-fine-tuned models such as MiniLM offer high accuracy and significantly faster inference times than large, general-purpose models. Model choice ultimately depends on practical requirements: prioritize the best model for maximal overall accuracy, or prefer the mixed-data configuration when balanced performance across classes is crucial.

6.2 Recommendations for Future Work.

Further research should focus on deeper analysis of the embedding space, exploring how receipt semantics are represented within the embedding manifold. Visualizations

and clustering techniques could help identify semantic overlaps and areas where the embedding quality can be improved. Additionally, incorporating a larger and more diverse set of real-world receipt data would likely enhance generalization and robustness, particularly for minority classes that are currently underrepresented. Such expanded datasets would facilitate more effective fine-tuning and model optimization, potentially driving further performance gains.

6.3 Limitations of the Study

Although the presented approach achieves strong overall performance, there are several notable limitations. First, the primary dataset used for training and evaluation is limited in size, comprising only 26 receipts and a total of 292 items. This relatively small and specific dataset restricts the diversity and representativeness of the training examples, potentially affecting the generalization capability of the models, especially for infrequent or rare categories.

Second, despite the introduction of GPT-generated data to mitigate class imbalance, significant disparity in class representation remains. The dominance of certain categories such as *Food* mirrors real-world scenarios but poses challenges to reliably classify minority categories, as synthetic data might not perfectly capture real-world variability and semantics. Lastly, the evaluation exclusively utilized Polish-language receipts. Consequently, the applicability and performance of the developed system in other linguistic or regional contexts remain unverified, limiting generalizability and transferability of these findings.

Addressing these issues in future research would further enhance the robustness and applicability of the developed models.

7 Conclusion

7.1 Summary of Findings

This research highlights several key findings regarding embedding methods and data augmentation for receipt classification tasks. Fine-tuning the Sentence-Transformer embedding model (`all-MiniLM-L6-v2`) exclusively on real-world receipt data resulted in the highest overall (micro) F1 score of 0.81, emphasizing the importance of domain-specific embedding adaptation. Incorporating GPT-generated synthetic data at the classifier training stage significantly improved generalization and particularly enhanced predictions for minority classes. However, directly fine-tuning embeddings on mixed GPT and real data slightly reduced overall accuracy, although it increased class-wise balance (macro-averaged F1 score of 0.77).

Despite its larger architecture, the pretrained Polish BERT model (`dkleczek/bert-base-polish-cased-v1`), used without task-specific fine-tuning, consistently underperformed, reinforcing the necessity of targeted, domain-specific model adaptation rather than relying solely on raw model size. Class imbalance was a consistent challenge: the dominant category *Food* consistently achieved high scores (F1 up to 0.88), while minority categories such as *Housing* and *Healthcare* showed improved performance with GPT augmentation but remained difficult due to inherent semantic variability and limited training samples.

Finally, practical considerations indicated that compact, domain-specific fine-tuned embedding models offered both higher accuracy and faster inference compared to large, generic transformer models, thus making them preferable for real-world applications. These results collectively emphasize the value of domain-specific fine-tuning, careful synthetic data augmentation, and efficiency considerations when deploying NLP-based classification pipelines.

7.2 Contributions to the Field

This thesis provides several important contributions to the area of Natural Language Processing and receipt-based product categorization. Firstly, it presents a specialized

end-to-end classification pipeline specifically tailored to Polish-language receipts, addressing a practical gap in existing personal finance management tools. Secondly, this research offers valuable insights into embedding-model strategies, demonstrating the significant advantages of domain-specific fine-tuning over using large pretrained models.

Additionally, this study thoroughly examines the role of GPT-generated synthetic data, clearly illustrating its effectiveness in addressing class imbalance issues within receipt classification tasks. By systematically benchmarking different embedding and classification approaches, the thesis provides a clear comparative analysis and establishes robust performance benchmarks for future research. Finally, by evaluating practical aspects such as computational efficiency and inference speed, the research delivers actionable recommendations for effective and efficient real-world implementations of NLP-based categorization systems.

8 Appendices

8.1 Example Receipt and Annotation

Figure 8.1 presents an example receipt image used as an input for the OCR and annotation process.

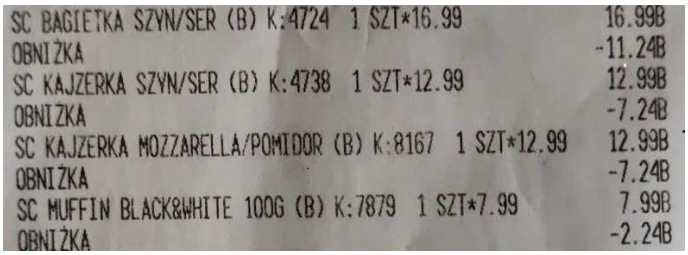

Figure 8.1. Sample receipt image

After applying OCR (using PaddleOCR), the textual output was manually annotated with category labels and associated costs. Below is an example of annotated receipt data in CSV format:

```
"SC BAGIETKASZYN/SER BK47241SZT*16.9S 16.99B",Food,16.99
"OBNIZKA -11.24B",Discount,11.24
"SCKAJZERKASZYN/SERBK:47381SZT*12.99 12.99B",Food,12.99
"OBNIZKA -7.248",Discount,7.24
"SC KAJZERKA MOZZARELLA/POMIDOR (B)K81671 SZT*12.99 12.99B",Food,12.99
"OBNIZKA -7.248",Discount,7.24
"SC MUFFIN BLACK&WHITE100G BK:78791SZT*7.99 7.99B",Food,7.99
"OBNIZKA -2.24B",Discount,2.24
```

This structured textual data serves as input for subsequent preprocessing, embedding generation, and classification steps detailed in earlier sections.

8.2 Implementation Examples

8.2.1 Text Preprocessing

The following example shows a text normalization function used after OCR extraction to reduce noise and standardize textual data:

Listing 8.1. Post-OCR text preprocessing

```
def preprocess_text(text):
    text = re.compile('[%s]' % re.escape(string.punctuation)).sub('␣',
    text = re.sub(r'\s+', '␣', text)
    text = re.sub(r'[^\w\s]', '', text.strip())
    text = re.sub(r'\d', '␣', text)
    text = re.sub(r'\s*\b[a–zA–Z]\b\s*', '␣', text).strip()
    return text
```

This preprocessing step standardizes the extracted OCR text by removing punctuation, numbers, single characters, and excessive whitespace, thus improving subsequent embedding quality.

8.2.2 OCR Pipeline

Below is an example of PaddleOCR implementation and extraction of preprocessed data from image files:

Listing 8.2. OCR class

```
class OCR:
    def __init__(self):
        self.paddleocr = PaddleOCR(
            use_angle_cls=True, lang='pl',
            rec_algorithm='CRNN', rec_char_type='pl'
        )

    def get_preprocessed_data(self, img_path):
        ocr_data = self.perform_ocr(img_path)
        preprocessed = []
        for line in ocr_data:
```

```
            for word_info in line:
                text = preprocess_text(word_info[1][0])
                cost = preprocess_cost(word_info[1][0])
                preprocessed.append((text, cost))
        return preprocessed
```

This OCR pipeline leverages PaddleOCR to extract textual content from images, applying preprocessing steps to ensure high-quality inputs for further analysis.

8.2.3 Sentence-Transformer Fine-tuning

The example below demonstrates how the Sentence-Transformer embedding model (`all-MiniLM-L6-v2`) was fine-tuned using contrastive loss to better capture domain-specific semantics:

Listing 8.3. Fine-tuning Sentence-Transformer embeddings

```
model.fit(
    train_objectives=[(train_dataloader, loss)],
    evaluator=evaluator,
    epochs=epochs,
    warmup_steps=warmup_steps,
    evaluation_steps=len(train_dataloader),
    output_path=path,
    save_best_model=True,
)
```

Fine-tuning was performed using labeled pairs of receipt texts to ensure embeddings accurately reflect semantic differences relevant to categorization tasks.

8.2.4 Hydra-driven Model Training Pipeline

The Hydra framework was utilized for structured experimentation, enabling easy parameter adjustments and reproducibility:

Listing 8.4. Training pipeline

```
@hydra.main(config_path="./conf", config_name="config")
def main(cfg):
    dataloader = Dataloader(Path(cfg.datapath))
```

```
    embedding_model = Embedding.load_mini_model(cfg.embeddingmodel.mod

    train_batch, test_batch, _ = dataloader.get_training_ready_data(
        embedding_model, split_size=cfg.split_size
    )

    model = XGBoost(**cfg.params)
    model.fit(train_batch['embeddings'], train_batch['category'])
    results = model.test_model(test_batch['embeddings'], test_batch['ca

    hydra.utils.log.info(f'Test F1 Score: {results["f1_score"]}')
```

Using Hydra streamlined experimentation, provided automated logging of parameters and results, and ensured experimental reproducibility.

8.3 Additional Figures and Tables

8.3.1 XGBoost Hyperparameters by Model Type

The hyperparameters used for the XGBoost classifier differ slightly depending on the embedding configuration. Below are the tuned values for each scenario.

Listing 8.5. XGBoost hyperparameters – BERT model

```
learning_rate: 0.3
max_depth: 5
n_estimators: 50
subsample: 0.8
colsample_bytree: 1.0
min_child_weight: 5
gamma: 0.1
reg_alpha: 0.0
reg_lambda: 0.0
```

Listing 8.6. XGBoost hyperparameters – Sentence-Transformer (GPT + real fine-tuned)

```
learning_rate: 0.2
max_depth: 7
```

```
n_estimators: 500
subsample: 0.5
colsample_bytree: 0.6
min_child_weight: 7
gamma: 0.0
reg_alpha: 0.0
reg_lambda: 0.0
```

Table 8.1. Full classification report – BERT model (out-of-the-box, no fine-tuning)

Class	**Precision**	**Recall**	**F1 Score**	**Support**
Beverages (0)	0.25	0.14	0.18	7
Discount (1)	1.00	0.67	0.80	6
Food (2)	0.78	0.89	0.83	55
Healthcare (3)	0.50	0.67	0.57	6
Housing (4)	0.00	0.00	0.00	4
Other (5)	0.86	0.60	0.71	10
Macro Average	0.56	0.49	0.51	88
Weighted Average	0.71	0.73	0.71	88
Overall Accuracy		0.7273		
Overall F1 Score		0.7073		

Bibliography

[1] Felipe Almeida i Geraldo Xexéo. *Word Embeddings: A Survey*. 2023. arXiv: 1901.09069 [cs.CL]. URL: https://arxiv.org/abs/1901.09069.

[2] Tianqi Chen i Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". W: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. ACM, sierp. 2016, s. 785–794. DOI: 10.1145/2939672.2939785. URL: http://dx.doi.org/10.1145/2939672.2939785.

[3] Jacob Devlin i in. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL]. URL: https://arxiv.org/abs/1810.04805.

[4] Yuning Du i in. "PP-OCR: A Practical Ultra Lightweight OCR System". W: *arXiv preprint arXiv:2009.09941* (2020).

[5] Amit Gupte i in. *Lights, Camera, Action! A Framework to Improve NLP Accuracy over OCR documents*. 2021. arXiv: 2108.02899 [cs.CL]. URL: https://arxiv.org/abs/2108.02899.

[6] Noman Islam, Zeeshan Islam i Nazia Noor. "A Survey on Optical Character Recognition System". W: *ITB Journal of Information and Communication Technology* (grud. 2016). DOI: 10.48550/arXiv.1710.05703.

[7] Daniel Jurafsky i James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3rd. Online manuscript released January 12, 2025. 2025. URL: https://web.stanford.edu/~jurafsky/slp3/.

[8] Minghui Liao i in. *Real-time Scene Text Detection with Differentiable Binarization*. 2019. arXiv: 1911.08947 [cs.CV]. URL: https://arxiv.org/abs/1911.08947.

[9] Tomas Mikolov i in. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL]. URL: https://arxiv.org/abs/1301.3781.

[10] Alexey Natekin i Alois Knoll. "Gradient Boosting Machines, A Tutorial". W: *Frontiers in Neurorobotics* 7 (2013), s. 21. DOI: 10.3389/fnbot.2013.00021.

[11] Nils Reimers i Iryna Gurevych. “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks”. W: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, list. 2019. URL: https://arxiv.org/abs/1908.10084.

[12] Nils Reimers i Iryna Gurevych. *sentence-transformers/all-MiniLM-L6-v2*. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2. Accessed: 2025-05-29. 2021.

[13] Baoguang Shi, Xiang Bai i Cong Yao. *An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition*. 2015. arXiv: 1507.05717 [cs.CV]. URL: https://arxiv.org/abs/1507.05717.

[14] Baoguang Shi, Xiang Bai i Cong Yao. “An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition”. W: *arXiv preprint arXiv:1511.04176* (2016). DOI: 10.48550/arXiv.1511.04176.

[15] Ray Smith. “An overview of the Tesseract OCR engine”. W: *Ninth International Conference on Document Analysis and Recognition (ICDAR)*. IEEE. 2007, s. 629–633.

[16] Ray Smith. “History and future of the Tesseract OCR engine”. W: *Document Recognition and Retrieval XX*. T. 8658. SPIE. 2013, s. 865802.

[17] Nishant Subramani i in. “A Survey of Deep Learning Approaches for OCR and Document Understanding”. W: *CoRR* abs/2011.13534 (2020). arXiv: 2011.13534. URL: https://arxiv.org/abs/2011.13534.

[18] Nishant Subramani i in. *A Survey of Deep Learning Approaches for OCR and Document Understanding*. 2021. arXiv: 2011.13534 [cs.CL]. URL: https://arxiv.org/abs/2011.13534.

Wyrażam zgodę na udostępnienie mojej pracy w czytelniach Biblioteki SGGW w tym w Archiwum Prac Dyplomowych SGGW.

.....
(czytelny podpis autora pracy)