Erasmus University Rotterdam

Erasmus School of Economics

Master Thesis

Business Analytics & Quantitative Marketing

Econometrics and Management Science

Classifying Documents using both Textual and Visual Features

19th August 2019

Author: Auke Zijlstra

Student Number: 387892

Abstract

With the increasing importance of Customer Due Diligence, financial institutions are forced to digitize and categorize their paper archives. Currently, document classification is primarily realized using textual information. This paper proposes to complement textual features with visual features, using a convolutional neural network and transfer learning. The proposed approach is tested on both a small real-world data set for a large Dutch bank, and on the larger academic RVL-CDIP data set. It is found that using the combined approach yields better classification performance than using only textual or visual features. For the RVL-CDIP data set the proposed method achieves state-of-the-art accuracy of 93.51%, exceeding previous results based on solely visual features. For the smaller real-world data set, the combined method scores marginally better than the benchmark set using only textual features, while being computationally much more expensive. Therefore, it is concluded that adding visual features using deep learning is a favorable approach to increase document classification performance, given that the data and computational resources are available.

Keywords: text classification, document image classification, feature combination, tf-idf, convolutional neural network

Supervisor: Prof. Dr. Ilker Birbil

Second Assessor:

This page is intentionally left blank

Contents

1 Introduction

As computers and electronic communication slowly started to emerge into everyday life, the concept of a 'paperless society' was introduced by Lancaster in 1978. A paperless society would be a society in which paper communication was replaced by digital (electronic) communication and storage. As a result, the need for paper would decrease and eventually 'digital' would completely replace 'paper'. 40 years later, the paperless society has still not materialized and our current information society is producing more and more information, of which a large part is still printed on paper.

However, even if all information would be stored digitally, it would be mostly encoded in natural language, either in English or any other language that is around. Although the technological possibilities and computing power have increased enormously over the past 40 years, the processing of natural language remains one of the more difficult tasks for a computer. One of the applications of Natural Language Processing (NLP) is the classification of text or documents into categories, describing the topic or type of a certain text or document. While document classification within the field of NLP has focused exclusively on the textual basis of documents, in practice there are more features that can be used for classification. Besides textual features, one of the most important types of document features are visual features, that is, the layout of the document that needs to be classified. This thesis focuses on the combination of textual and visual document features, within the context of document classification.

The outline of the problem that this thesis intends to investigate has been supplied by Rabobank and formed the basis for an internship. The goal of this internship was to classify a set of financial documents of Rabobank's subsidiary De Lage Landen (DLL) into approximately 20 (given) categories, based on what type of document it is. This (re)classification of documents is mandatory for the Rabobank, based on the guidelines issued by the European Central Bank (ECB). The classification will be done based on both textual and visual features, extracted from the documents. This leads to the following research question:

What is the best method to classify financial documents using both textual and visual features?

The above research question is chosen to be broad, as there are many ways to determine what the best method is to classify documents. In doing so, this thesis will not focus on classification accuracy alone. Other factors that are taken into consideration are robustness, computational efficiency, the amount of training data needed and the possibility to transfer models between domains/applications. Next to that, this thesis does not focus solely on textual features or visual features, but investigates the combination of both fields for classification purposes.

The motivation for this research question (and the considered factors) mainly follows from the application at hand. While good classification performance in terms of accuracy is important, from a business perspective it is also important that this performance is consistent over different document types and is not overly dependent on document (image) quality. In addition, the computational efficiency of the used method is important. Although computational power

has increased enormously over time (and has relatively become cheaper), the size of the problem (initially 1.3 million documents, 300 million in total) makes it necessary to take computational efficiency into account. Moreover, the amount of labelled training data differs per application, but is often relatively small compared to the complete data set. Therefore, it is important to take the amount of labeled training data needed for different methods into account. For example, it could be that good classification speed and performance can be obtained by means of a Neural Network, but that this method is less feasible because the amount of training data is greater than (generally) available. Lastly, besides needing relatively little training data, the possibility to transfer models between domains or specific applications shortens the implementation time for future projects.

The contributions of this work are twofold:

1. While the majority of the academic literature within the field of document classification focuses either on textual features or visual features, there is a gap in research that focuses on the combination of both fields. This work contributes to academic literature by looking specifically at the combination of both approaches. Moreover, the perspective of the research question is wider than only accuracy, contributing to the existing academic literature that often primarily focuses on classification accuracy metrics.

2. The second contribution of this work lies within the practical application for the Rabobank. Currently the bank does not have a technical solution for the document classification problem at hand, besides using manual labor. However, the problem size makes this an unfeasible and costly solution. This work contributes to the problem by providing them with an automated solution to perform document classification based on textual and visual document features.

The remainder of this thesis is structured as follows: Section 2 covers the relevant academic literature concerning textual and visual features for document classification. Section 3 covers the data used in this thesis. Afterwards, Section 4 covers the methodology, first concerning the textual classification and then concerning the visual classification. This is followed by Section 5, where the results for both data sets are presented. Lastly, Section 6 presents the main conclusions of this thesis, and discusses limitations and suggestions for further research.

2 Literature Review

This section describes the relevant literature related to the research question. First it describes the literature regarding textual features, secondly it describes the literature regarding visual features. Lastly, the literature regarding the combination of both type of features is covered.

2.1 Textual Features

At the core of Natural Language Processing (NLP) lies the problem of how information is extracted from text. Human language can be considered highly efficient, as humans are able to convey a lot of information using relatively short amounts of text. However, the disadvantage of this efficiency is that human language is often highly ambiguous and relates back to information that was either given previously in the same text, or is presumed to be already available to the reader. Over time, linguists, later followed by computer scientists, have come up with a multitude of different ways to deal with this information retrieval problem, often specifically focused towards the task at hand. Within the process of document classification, often the steps shown in Figure 1 can be distinguished (Sumathy & Chidambaram, 2013). The following subsections (briefly) look into the different steps in the classification process.

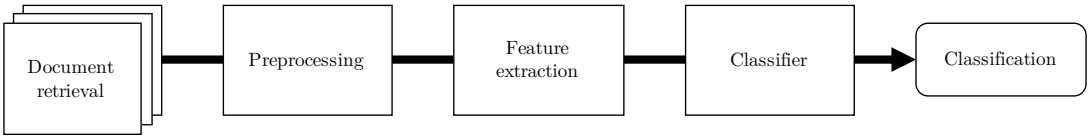


Figure 1: Schematic overview document classification steps

2.1.1 Document retrieval

The document retrieval steps are largely dependent on the document source. In the academic literature, there are multiple standard textual data sets available, which are often relatively clean and well structured, such as the twenty newsgroup data set (Lang, 1995) or the IMDB large movie review data set (Maas, Daly, Pham, Huang, Ng & Potts, 2011). When using a new data set, the origin of the text is important to take into account. Nowadays, text is often already digitally available, for example because the text was scraped from the internet.

However, when this is not the case, the first step towards document classification is to make the textual contents of the document digitally available by performing optical character recognition (OCR). The quality of OCR is largely dependent on the quality of the input images, but also on the language used, typesetting and layout used within the document (Govindan & Shivaprasad, 1990). Although character recognition performance can be improved by preprocessing input images, some typical OCR errors are generally found in the digital output (e.g. 'i' being recognized as 'l'). Taghva, Borsack & Condit (1996) discuss that this often has no material effect for general purposes, but could raise potential problems for documents with particular features

such as tabular data. To perform OCR, open source OCR software is available supporting more than 100 languages, with error rates as low as 1% (Smith, 2007).

2.1.2 Preprocessing of text

A second step in the text classification process is the preprocessing of the textual input. The goal of this step is to reduce and clean the textual input, by removing noisy and non-informative parts. However, what is considered to be noise or non-informative depends on the problem setting and the feature extraction method used.

Uysal & Gunal (2014) provide an overview of commonly used preprocessing steps when performing text classification and investigate the impact each separate step has on the classification performance. They include: (1) lowercasing, (2) removing stops words (words that carry little meaning and/or occur frequently such as pronouns prepositions and conjunctions), (3) tokenizing (splitting text into seperate terms) and (4) stemming (reducing words to their stem). In their emprical evaluation, Uysal & Gunal (2014) find that lowercasing, tokenizing and stemming positively influences classification results, confirming previous literature. Interestingly, for stop words removal they find a negative effect, while previous literature generally showed positive effects. A possible negative effect of tokenizing is that words consisting of multiple terms (e.g. "New York") are split into seperate tokens. Therefore, not only unigrams (single tokens), but also bigrams and higher order N-grams can be included in the list of tokens.

2.1.3 Feature extraction

After preprocessing, the (tokenized) words (hereafter referred to as terms) must be transformed to a numeric representation, that can afterwards be used in a classifier. This step is known as feature extraction. Although there exist multiple ways in which a piece of text can be transformed into a numerical representation, this thesis focuses on two different methods: the sparse representation of text and the dense representation of text.

The sparse representation of text is based on the Vector Space Model (VSM) of Salton, Wong & Yang (1975), which is based on the *bag of words hypothesis*. This hypothesis states that a document can be represented as the bag of words (BOW) that make up the document. In the setting of document classification, this implies that the frequency of words in a document indicates to what class a document belongs. In the VSM, a set of document is transformed into a document-term matrix. In this matrix, each document forms a row-vector, with each term in the vocabulary represented as a separate column. Consequently, each value x_{ij} in document-term matrix $\boldsymbol{X}$ indicates how often term j occurs in document i. In general, the resulting document representation is a very sparse vector that contains mainly zeros, since most terms do not occur in a given document.

In the simple VSM as described above, a document is represented only as a BOW, not taking into account the relative importance of different terms in different documents. Therefore, it is

common to re-weight the document-term matrix by using a form of *term frequency-inverse document frequency* (tf-idf) re-weighting (Salton & Buckley, 1988). The idea of tf-idf is based on counting terms within and across documents, based on the hypothesis that terms that occur more frequently across documents are less informative features. Although the order in which words occur in a document is lost when representing a document using the VSM, in practice tf-idf is widely used and performs often reasonably good (Sebastiani, 2002; Zhang, Zhao & LeCun, 2015). Therefore, tf-idf is used as the text feature extraction algorithm in this thesis.

Besides that the VSM does not take into account the word ordering, it also ignores the structuring of the input document. For example, text contained in the header of a document is equally important (before re-weighting) as text in the body of a document, although it might be more important with regards to the classification of the said document. This motivated Bratko & Filipič (2006) to investigate three possible methods of including (predefined) structural information in the context of text classification, being: tagging, splitting and stacking. They find that stacking to be the best way to include structural information, clearly outperforming plain text classification. Although the data used in this thesis lacks a predefined structure, the findings of Bratko & Filipič (2006) show that including structural information improves classification performance. Therefore, this thesis looks into segmented textual feature extraction as a way to include structural document information.

Representing text in a dense vector, as opposed to the sparse representation described earlier, is a relatively new concept. Although the idea has been around for quite some time (Hinton, McClelland, Rumelhart & others, 1984), applying this concept really took off after the publications by Collobert, Weston, Bottou, Karlen, Kavukcuoglu & Kuksa (2011) and Mikolov, Chen, Corrado & Dean (2013). When using distributed representations at a word level, every word is encoded into a continuous vector (known as word vector or word embedding) in a low dimensional vector space, in such a way that word's semantic and syntactic similarity (which can be thought of as word meaning) is encoded within the vector representation. Low dimensional in this case refers to between 50 and 1000 dimensions, whereas high dimensional refers to a dimensionality equal to the amount of words in the vocabulary.

Mikolov et al. (2013) show that by using their Neural Network model architecture, they are able to obtain word vectors that encode semantic and syntactic information in such a way that they are able to perform mathematical operations on words. For example, they find that the vector("$King$")$-$vector("Man")$+$vector("$Woman$") gives a resulting vector that lies closest to the vector("$Queen$") in the vector space. The authors called their improved model *word2vec* and released the source code to the (academic) community, resulting in a lot of follow up research.

Another influential paper in the word embedding literature followed a few years later, when Joulin, Grave, Bojanowski & Mikolov (2017) published their *fastText* model. The main novelty of the their model was using subword N-grams (combinations of 3-6 characters that 'build' up a word), not only enabling them to calculate embeddings for words that are out of vocabulary

(OOV) but also improving the embeddings for rare words. OOV here means that the word is not in the corpus of words previously seen by the model and therefore no word embedding is available. In this thesis a large set of financial documents is classified, containing relatively more words that are rare outside of the financial domain. Besides that, calculating embeddings for previously unseen words as done by the *fastText* model is especially useful for languages that share a lot of subword N-grams between words (often seen in compound words), such as Dutch and German (Bojanowski, Grave, Joulin & Mikolov, 2017). For these languages, it is hypothesized that the resulting OOV word embeddings will capture relatively much of the word meaning.

A large part of the work on textual embeddings has focused on words as smallest textual *units* to train distributed embeddings on. However, already shortly after the seminal work of Mikolov, Sutskever, Chen, Corrado & Dean (2013) on word embeddings, further research was conducted on how to extend this idea to larger text bodies such as paragraphs or complete documents. A simple approach to do so is by taking a average of the word embeddings to create document embeddings, which can consequently be used as input for a classification algorithm. Intuitively, this would result in a document embedding that describes the 'topic' of a document, based on the (topic-specific) words in the document. However, simple word embedding averaging does not take into account the word-order and word-context, but simply represents a document as a 'bag of word embeddings'. Thus, common words could have a disproportionately large influence on the created document embeddings, resulting in document embeddings that all converge to a common centroid.

With this in mind, the findings of Zhang et al. (2015) are especially relevant, who show that a *bag of means* approach yields consistently bad classification results. Moreover, they conclude that N-gram tf-idf methods remain the best approach in data sets up to several hundreds of thousand observations, while neural network based embedding techniques perform better in larger data sets. As a preliminary study, we have tested several dense embeddings techniques on the Rabobank data set, and came to similar conclusions as Zhang et al. (2015). Based on this result, this thesis focuses on using tf-idf re-weighted VSM sparse embedding techniques for the textual features, limiting the scope towards the more recent neural network based dense embeddings techniques.

2.1.4 Classifiers

Within the text classification literature, several different classification algorithms have been applied. Yang & Liu (1999) provide a comparison of five often used classification algorithms: Support Vector Machines (SVM), a k-Nearest neighbor (kNN) classifier, the Linear Leastsquares Fit (LLSF) mapping, a Neural Network approach and a Naive Bayes (NB) classifier. They find that the first three classification algorithms significantly outperform the last two, especially if less training data is available.

Moreover, Joachims (1998) has already shown that SVMs perform well in the context of

text categorization, theoretically supporting his findings based on the high dimensional feature space, containing relatively few irrelevant features and the sparseness of the instance vectors. A later study by Dumais, Platt, Heckerman & Sahami (1998) confirms the findings by Joachims (1998), pointing out that their usage of the simpler linear kernel performs even better than the radial basis function kernel used by Joachims (1998), with the advantage of being simpler and more efficient.

Based on these works, this thesis uses a SVM with linear kernel as base classifier. However, with regards to combining features, a disadvantage of the SVM is that is does not output a vector of class probabilities. Platt & others (1999) have shown that fitting a sigmoid function to the output of the SVM can help overcome this problem, but this would imply an extra model fitting step. Therefore, this thesis uses the often quite similarly performing logistic regression as second classifier. Because the logistic regression optimizes a logistic loss function (compared to the hinge loss in SVMs), it is expected that the logistic regression will be more sensitive to outliers.

2.2 Visual features

Besides including textual features, as described in Section 2.1, it is also of interest to include visual features from the documents that need to be classified. The main reason for this is that the formatting of the documents follows to a great degree from the type of document. For example, a letter usually contains one or two addresses on the top of the first page, followed by a salutation and one or more (short) text paragraphs. Finally, a letter is usually ended by a greeting and/or signature, followed by a name. Even if the text within a letter would be illegible, one can usually recognize the document type from the formatting described above. Similar features could be thought of for different types of financial documents, such as invoices, bank guarantees or contracts. Moreover, as the document set at hand originates from one company (the Rabobank), many documents follow more or less the same standard company template, which could yield discriminatory power between different document classes.

Within the field of information retrieval, classifying documents without using textual information has been a research topic for quite some time. Chen & Blostein (2007) provide a survey of the literature on document image classification at the time, structuring their survey along three components: (i) the problem statement, (ii) the classifier architecture, (iii) and the performance evaluation. They show that there is a wide variety in methods used, following from the document space, document classes, the way in which document features are represented and the type of classifier used. Although their survey does include some statistical pattern classification techniques such as nearest neighbor, decision trees and neural networks, at the time these models were not used in the majority of the academic applications.

In the past ten years, just as in the field of natural language processing, neural network models have been increasingly used in visual document classification. Following other computer vision literature (e.g. object recognition), Harley, Ufkes & Derpanis (2015) apply a convolutional

neural network approach to document image classification and retrieval, showing that their Machine Learning approach outperforms 'hand-crafted' alternatives by a substantial margin. Moreover, they released a large new data set called RVL-CDIP [1], consisting of 400,000 grayscale document images, covering a total of 16 different document classes.

Based on this data set, further model improvements have been suggested by various authors (Tensmeyer & Martinez, 2017; Afzal, Kölsch, Ahmed & Liwicki, 2017), with the current best performing model to our knowledge being Das, Roy, Bhattacharya & Parui (2018). Interestingly, the model proposed by Das et al. (2018) makes use of inter-domain transfer learning. This is done by using freely available initial model weights that were obtained by Simonyan & Zisserman (2014), who trained their model on the ImageNet (Deng, Dong, Socher, Li, Li & Fei-Fei, 2009) data set, commonly used in the domain of object classification.

Similarly as in the field of NLP, it is shown that using initial model weights (word embeddings respectively) obtained from a different domain yield better classification performance than randomly initializing model weights. This thesis uses the methodology of Das et al. (2018) as inspiration for extracting and modelling the visual features.

2.3 Combination of features

Instead of looking only at textual features as in Section 2.1 or only at visual features as in Section 2.2, taking a combination of features could yield better classification performance. Within the academic literature, combining features from different modalities is commonly referred to as a multimodal approach. Although relatively less academic work has been done on the combination of features than on the the individual type of features, there has been some previous literature.

Within the field of medical imaging, relevant academic work has been done as submissions for the ImageCLEF cross-language image retrieval track of the Cross Language Evaluation Forum (CLEF) (Villegas, Müller, Gilbert, Piras, Wang, Mikolajczyk, de Herrera, Bromuri, Amin, Mohammed & others, 2015). In the 2015 edition of the ImageCLEF medical classification track, the winning submissions used both textual and visual features, outperforming submissions based on single modalities by a substantial margin (Pelka & Friedrich, 2015). In their submission, the authors used feature-level fusion (Shan, Gong & McOwan, 2007), combining the features (with reduced dimensionality using principal component analysis) from textual and visual input in a 'joint vector', before applying a linear SVM classifier.

Another possibility to fuse modalities is decision-level fusion; modelling and classifying each modality independently, combining unimodal results at the end. The exact way in which unimodal results are combined can differ, but usually simple operations such as majority votes, sums, products or statistical weighting are used. According to Poria, Cambria, Howard, Huang & Hussain (2016), some earlier studies prefer decision-level fusion over feature-level fusion, because in the former errors from different classifiers are uncorrelated. Moreover, the former

[1] Data set information can be found on the RVL-CDIP webpage: http://www.cs.cmu.edu/ aharley/rvl-cdip/ (Last access date: August 12, 2019)

method gives more flexibility regarding the methodology applied to the different modalities.

A comparison between both methods of fusing modalities was made by Poria et al. (2016), when combining textual, visual and audio modalities to perform sentiment analysis on Youtube videos. It was shown that the proposed multimodal system outperformed all unimodal state-of-the-art systems by more than 20%, achieving a total accuracy of nearly 80%. This accuracy was achieved using feature-level fusion, although the difference between both fusion methods was only 3%. Based on these papers, this thesis uses decision-level fusion to combine textual and visual modalities.

3 Data

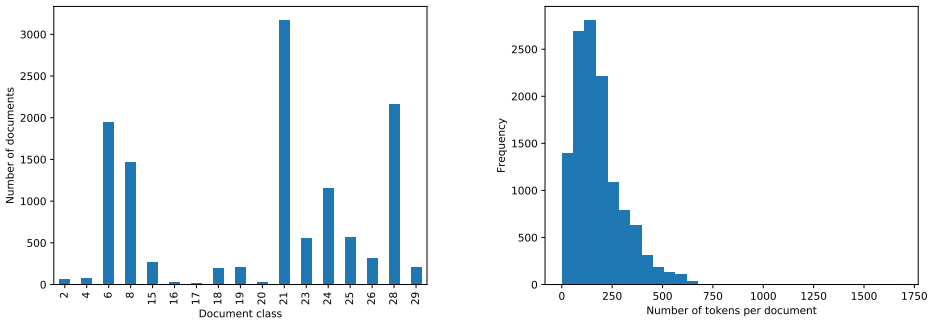
In this thesis, we use two data sets: one data set originating from the Rabobank and one academic data set named RVL-CDIP, first put forward by Harley et al. (2015). This section describes both data sets, followed by a brief discussion on the way the data is used for training, validation and testing.

Rabobank

The data set corresponding to the classification problem at the Rabobank is part of a larger data set, consisting of over a million financial documents. The document types are (among others) lease contracts, lease agreements, checklists, invoices, correspondence and other types of financial documents. Parts of the documents are hypothesized to follow a more or less predefined document structure, therefore it should be possible to perform *visual* categorization. Moreover, the textual content of the documents is hypothesized to be of such nature that it should be possible to perform further *textual* categorization, for example regarding the exact type of financing agreement. The documents originate from De Lage Landen (DLL) which is a vendor finance company owned by the Rabobank Group.

The raw input files are scanned images, on which optical character recognition (OCR) is applied. The individual files consist of scans containing from one page (short email, several kilobytes file size) to 300+ pages (multiple contract with appendices, 100 megabytes file size). A data characteristic that complicates the classification problem, is that a single input file usually contains several documents, which have been scanned together. Therefore, the input files are split page-wise and consequently categorized per page. This implies that information is lost, because the page ordering is not taken into account. Moreover, the 'prior' knowledge that a page has a higher probability to belong to the same category as the preceding page (i.e. two pages belonging to the same document) is not taken into account.

From the larger data set, a subset has been labeled by business users. This subset contains in total $12,462$ labeled document pages, spanning 23 categories. The data set is highly unbalanced, which poses a problem when splitting the data set into train, validation and test sets. Therefore, the categories with less than 10 observations are removed, bringing the total

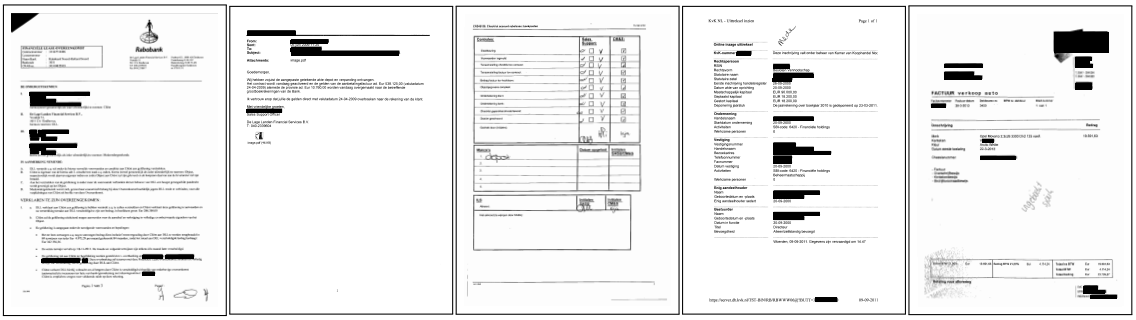


(a) Document class distribution (b) Textual length (of tokens) histogram

Figure 2

included observations to 12,440 observations in 17 categories. Figure 2a shows the distribution of the observations over the categories. The resulting data set contains about five large classes (> 1000 observations), seven medium sized classes ($100 - 1000$ observations) and five more small classes (< 100 observations).

Moreover, Figure 2b shows the distribution of the document length, measured as the number of found tokens, for the different documents. The majority of the documents includes about 200 tokens (words) with almost no documents having more than 750 tokens. Lastly, Figure 3 shows five (redacted) example documents that are representative for their respective document classes. Clearly, the document structure differs vastly between the classes, indicating that visual categorization is something to further investigate.



(a) Lease Contract (b) Correspondence (c) Checklist (d) KvK extract (e) Invoice

Figure 3: Example documents for several classes

RVL-CDIP

The second data set that will be used is the *RVL-CDIP* data set as gathered by Harley et al. (2015), from the larger IIT CDIP Test Collection (Lewis, Agam, Argamon, Frieder, Grossman & Heard, 2006). This larger data set contains document images (scans), which have been collected from publicly available document regarding lawsuits against a series of American

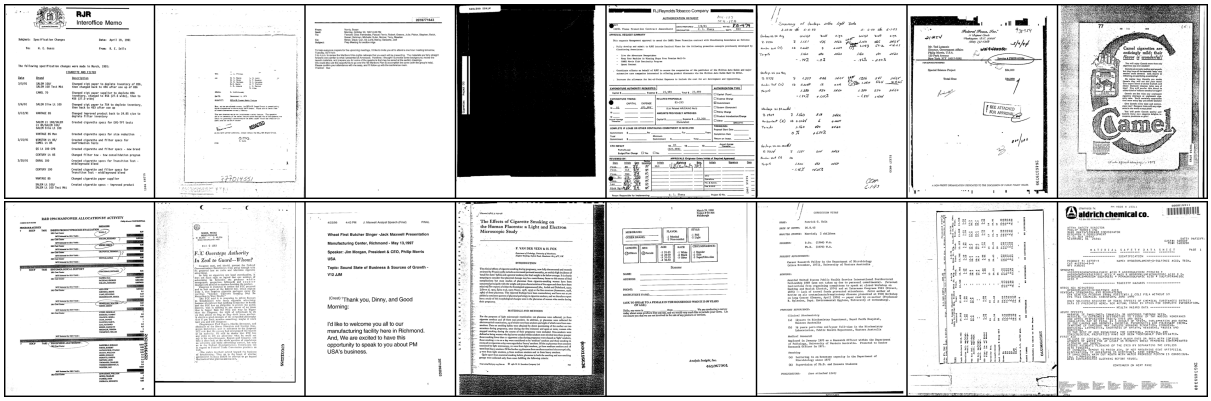


Figure 4: Example documents RVL-CDIP data set

tobacco companies. In the original IIT CDIP Test Collection data set, more than seven million documents have been included, in multiple unevenly distributed categories. However, in this data set, not all documents have been labeled (correctly).

From the IIT CDIP Test Collection Harley et al. (2015) gathered 400,000 labeled gray-scale document images, evenly distributed over 16 categories. Harley et al. chose the included categories based on earlier work on document categorization, taking into account the range of categories included in an earlier data set (SmallTobacco) based on the same IIT CDIP Test Collection. Moreover, they have restricted their sample selection to well represented categories, based on the knowledge that CNNs perform well with large data sets, i.e. when a large training sample is available. The categories that have been included in the RVL-CDIP data set are: "letter", "memo", "email", "filefolder", "form", "handwritten", "invoice", "advertisement" (first row), "budget", "news article", "presentation", "scientific", "publication", "questionnaire", "resume", "scientific report" and "specification" (second row). An example for each category is shown in Figure 4. Because the original IIT CDIP Test Collection data set sometimes had multiple tags per document, it could be that the final categories in the RVL-CDIP data set are not perfectly distinct, as only one tag per document is present in the current data set.

Based on the document id's present in the RVL-CDIP data set, we have retrieved the OCR text output from the IIT CDIP Test Collection, in order to test the textual classification performance. Unfortunately, this was not possible for the complete document set, resulting in 398,010 documents for which both the visual and the textual content is available. Consequently, for 1,990 documents no text could be found. For these documents, the textual input has been set to null. Next to that, for 11,356 documents a match could be made, although no text was available in the IIT CDIP Test Collection. Therefore, a total of 13,346 documents are included as not containing text. Besides that, for one document a corrupt document image is included. This document is therefore excluded, leading to a total test set size of 39,999 documents.

Data splitting

In this research, multiple modelling techniques are used, compared and combined in order to find the best modelling technique for the problem at hand. However, doing so is not trivial. Ultimately, we like to know how our model performs on unseen data. In other words, we like to estimate the generalization performance of the proposed models. For large data sets, this is usually by done by randomly splitting the data set into three parts: train, validation and test sets. If the observations are independent and identically distributed, and if the total number of observations n becomes large, this gives an unbiased estimation of the generalization performance of the model (Raschka, 2018). Therefore, this approach is taken for the RVL-CDIP data set, using the already predefined data splits introduced by Harley et al. (2015).

For smaller data sets, splitting the data set into three different parts is a less ideal situation, as the individual parts (train, validate and test set) will have relatively little observations. This could introduce a pessimistic model bias, as the model might not have reached its *capacity* (Raschka, 2018). In other words, the model would have performed better in case there was more training data available. In order to minimize this problem, we split the Rabobank data set into only two parts, using 80% of the data set as train data and the remaining 20% as test data. In order to take the class imbalances into account, splitting is done in a stratified fashion. This implies that the class proportions are similar in both the training and test set. The different models are trained on the train set and afterwards predictions are made for the unseen test set. In order to estimate the model performance, the predictions are then compared to the true labels in the test set, which have not been seen by the model before.

Next to comparing different modelling techniques, we also try to optimize the different models by selecting the optimal hyperparameters (tuning) for our models. This is done by training a model with different sets of hyperparameters and evaluating the model performance on an independent validation set. However, for this the test set cannot be used, as this would result in optimizing the model towards the test set, consequently positively overestimating the generalization performance. Hence, for the hyperparameter tuning k-fold cross validation is used.

The idea of k-fold cross validation is to randomly split the data set into k folds, train the model on $k-1$ folds and then validate the model on the remaining hold-out fold. This is done k times, each time holding out a different sample, as shown in Figure 5. Afterwards, the average performance of the model on the k hold-out sets is considered to be the best estimate of the model's generalization performance, to select the optimal set of hyperparameters. Having selected the optimal set of hyperparameters, the model is refitted on the entire train set and scored on the hold-out test set. Unless otherwise specified, this thesis uses $k = 5$ as the number of folds.

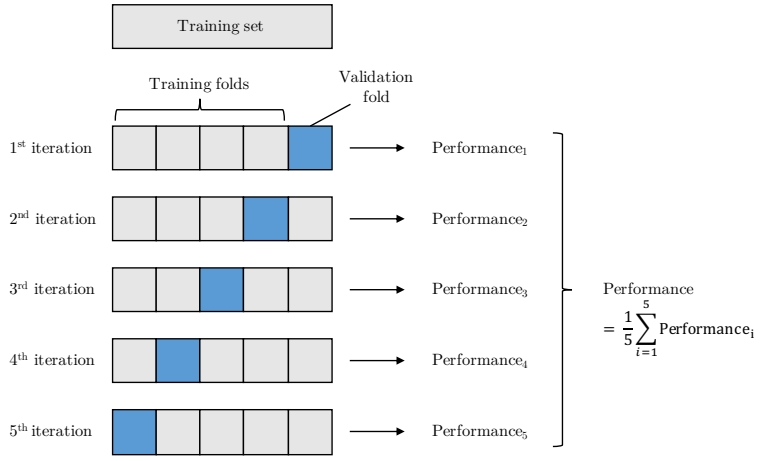


Figure 5: Schematic overview of 5-fold cross validation,
based on figure in Raschka (2018, p. 25)

4 Methodology

This section further elaborates on the methodology used to classify the different data sets. First, the methodology for the textual feature classification is explained, covering feature extraction, hyperparameters and classification algorithms. Secondly, the methodology regarding the visual features is covered, explaining the main concepts of classification using a convolutional neural network. Afterwards, we discuss how we combine both the textual and the visual features in one classification model, and how this combined model is trained. The last part of this section covers the evaluation measures that are used to compare the different models.

4.1 Textual features

This section covers the methodology related to the textual features. Based on the literature review and preliminary research, a sparse textual feature representation is chosen. After explaining how the textual features are extracted, hyperparameter tuning is covered, followed by the methodology of the classification algorithms used on the extracted features.

4.1.1 Feature extraction

The textual features are generated using *Term Frequency-Inverse Document Frequency* (tf-idf). Tf-idf is a numeric method that intuitively gives each word in a document a score, that reflects the relative importance of a word in a corpus. The tf-idf score is proportional to the number of times a word is found in a document and is corrected for the number of documents that contain the word. This helps to down weight words that appear more frequently in the whole corpus and are therefore hypothesized to be less informative for a specific document.

Mathematically, the tf-idf score for a term t in document d from corpus D can be calculated

as follows:

$$\text{tf-idf}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D), \tag{1}$$

where the term frequency $\text{tf}(t, d)$ is either the raw count of the number of times that term t appears in document d, denoted by $f_{t,d}$, or the logarithmically scaled version:

$$\text{tf}(t, d) = \log(1 + f_{t,d}). \tag{2}$$

The inverse document frequency $\text{idf}(t, D)$ is calculated by logarithmically scaling the ratio between the the total number of documents N and the number of documents containing the term, denoted by $|d \in D : t \in d|$:

$$\text{idf}(t, D) = \log \frac{N}{1 + |\{d \in D : t \in d\}|}. \tag{3}$$

Here the denominator is adjusted by adding one, in order to prevent division-by-zero if the term is not in the corpus. Moreover, both the term frequency and the inverse document frequency are scaled by taking the logarithm, to reduce the influence of terms that occur frequently.

Before applying tf-idf, input documents are preprocessed by removing stopwords, lower casing all terms, tokenizing and stemming, which is done using the NLTK library (Loper & Bird, 2002) [2]. This is done using the language settings corresponding to the document language, i.e. Dutch for Rabobank and English for RVL-CDIP documents.

4.1.2 Hyperparameter tuning

Besides applying the preprocessing steps, there are several hyperparameters that have to be set by the researcher relating to the textual feature extraction. In order to select these hyperparameters, a 5-fold cross validated grid-search is used. The hyperparameter values that will be considered for the tf-idf feature extraction are:

- Number of N-grams to be included: $\{(1, 1), (1, 2), (1, 3), (1, 4)\}$
- Minimum document frequency for terms in corpus: $\{2, 5, 10, 20\}$
- Maximum document frequency for terms in corpus: $\{0.5, 0.75, 1.0\}$
- Maximum number of features to be included: $\{1 \times 10^4, 2 \times 10^4, 3 \times 10^4, \text{no limit}\}$
- Tf-idf normalization: $\{l_1, l_2\}$
- Term frequency: {raw count, logarithmically scaled count}

One of the preprocessing steps is splitting the input text into terms, known as tokenization. Tokenization is done by using spaces and punctuation marks as delimiters. One of the negative effects of this process is that words consisting of multiple terms (e.g. "New York") are not taken into account. Therefore, not only unigrams (single tokens), but also bigrams and higher order N-grams (up till the 4th order) are included as terms in the tf-idf model. Because of the informational value of multi-word terms, it is hypothesized that including higher order N-grams will improve model performance.

[2] Using the `word_tokenize` function and the `SnowballStemmer`

Next to removing stop words, terms are removed if they appear very little or very frequently in the corpus. This is done by experimenting with multiple minimum document frequencies (absolute frequencies) and maximum document frequencies (relative frequencies) per term. Because terms that appear very rarely are hypothesized to be OCR artifacts, and terms that appear very frequent are hypothesized to be less informative, it is expected that this also improves model performance. Next to that, only the N features with the highest tf-idf scores are included, where initially four different values of N are considered.

The last two hyperparameters are related to the technical implementation of the tf-idf algorithm, and can both be useful to decrease the influence of frequently occurring terms. Either l_2 normalization is applied, making the sum of squares of the vector elements equal to one, or l_1 normalization is applied, setting the sum of absolute values of the vector elements equal to one. Both methods ensure that each output row has unit norm. Next to that, the term frequency is either taken as raw count or as logarithmically scaled count.

4.1.3 Classifiers

After the feature extraction using tf-idf, a supervised learning method is used as classification algorithm. This section describes the methodological aspects of both the (linear) Support Vector Machine (SVM) and the Logistic Regression (LR).

Support Vector Machine: The first supervised learning algorithm that is used is the SVM algorithm, as first proposed by Cortes & Vapnik (1995). The goal of the SVM algorithm is to construct a hyperplane through the data set that distinctly classifies the data points, i.e. that all points on one side of the hyperplane belong to the same class. After the hyperplane is constructed, a (non-probabilistic) prediction can be made for a new observation, by checking where the observation lies in the high-dimensional hyperspace. There could be many possible hyperplanes that separate the two classes of data points, but the 'optimal' hyperplane is the plane with the maximum margin. This means that hyperplane is chosen such that the distance between the nearest training-data observations (of any of the two classes; called the 'support vectors') and the plane is maximized.

In practice, a data set is usually not linearly separable and therefore a hyperplane cannot be constructed. This problem is solved by using a so-called kernel function, that has to be specified by the researcher. The kernel function maps the original data into a high-dimensional feature space. It follows that an optimal hyperplane can always be constructed in this feature space, resulting in a global optimal solution. This thesis uses a linear kernel, because of its good computation performance and because it was found that a linear kernel is usually sufficient for text classification problems (Yang & Liu, 1999). Afterwards, when turning towards feature combination, this thesis will also use the Radial Basis Function (RBF) kernel, for which Coussement & Van den Poel (2008) state several advantages over other possible kernel functions. Using the RBF kernel, it can be seen whether a non-linear kernel adds to the categorization performance

of the SVM when combining textual and visual features.

Mathematically, the SVM algorithm can be formulated as follows, assuming a binary data set that is linearly separable. Let y_i denote the classification variable, such that $y_i \in \{-1, 1\}$ with $\mathbf{x}_i \in \mathtt{R}^n$ the corresponding vector of explanatory variables for observation $i = 1, 2, \ldots, N$ and n the dimension of the training data space. Because the training data is linearly separable, for every observation in the training set the following equations must hold:

$$\mathbf{w} \cdot \mathbf{x}_i + b \leq -1, \qquad \text{when } y_i = -1; \tag{4}$$

$$\mathbf{w} \cdot \mathbf{x}_i + b \geq 1, \qquad \text{when } y_i = 1 \tag{5}$$

with $\mathbf{w}$ being a weight vector and b a scalar. Rewriting and combining these equations gives:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1. \qquad \forall i \tag{6}$$

Equation (6) can be interpreted as finding two parallel boundaries at each side of the separating hyperplane $\mathbf{w} \cdot \mathbf{x}_i + b = 0$, with the 'support vectors' lying at the boundary. Because finding the optimal hyperplane implies maximizing the distance to these 'support vectors', the margin width between the boundaries $\frac{2}{||\mathbf{w}||}$ needs to be maximized. From this the following (constrained) minimization problem can be formulated:

$$\begin{aligned} &\text{minimize} \quad \frac{1}{2}||\mathbf{w}||^2 \\ &\text{subject to} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 \end{aligned} \tag{7}$$

which can be solved using a Lagrangian function. By formulating a Lagrangian, solving the first order conditions and substituting back into the Lagrangian, the following expression can be obtained:

$$\begin{aligned} &\text{maximize} \quad L(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \\ &\text{subject to} \quad \alpha_i \geq 0 \text{ with } i = 1, 2, \ldots, N \text{ and } \sum_i \alpha_i y_i = 0, \end{aligned} \tag{8}$$

where the Lagrangian multipliers are denoted by α. However, in practice the data used will often not be linearly separable. Therefore a non-linear transformation Θ can be used to map $\mathbf{x}_i$ from its original space X to the higher dimensional feature space X', with $\Theta(\mathbf{x}_i)$ being the resulting value of observation x_i in the feature space X'. One would expect that this mapping to higher dimensions would give rise to computational complexity, however this is not the case. As can be seen in the maximization criterion of (8), only the dot product of $\mathbf{x}_i$ and $\mathbf{x}_j$ has to be calculated and consequently, only the inner product kernel has to be calculated in the feature space.

In the above formulation, non-linearly separable data was handled by using the kernel trick. Now imagine the case where you observe a data set with two classes that are pretty much separated, except for a small group of points that violate the linear separability criterion. Besides using a non-linear kernel, this situation can be handled by introducing positive slack variables

ϵ_i (relaxing the stiff condition of linear separability) in equations (4) and (5). This results in the following adjusted minimization criterion:

$$\begin{aligned} &\text{minimize} \quad \frac{1}{2}||\mathbf{w}||^2 + C\sum_i \epsilon_i \\ &\text{subject to} \quad y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \epsilon_i \text{ and } \epsilon_i \geq 0, \end{aligned} \tag{9}$$

with C being a cost parameter, penalizing the error terms. The resulting optimization problem, that will be used in this thesis is:

$$\begin{aligned} &\text{maximize} \quad L(\alpha) = \sum_i \alpha_i - \frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j y_i y_j \Theta(\mathbf{x}_i)\cdot\Theta(\mathbf{x}_j) \\ &\text{subject to} \quad 0 \leq \alpha_i \leq C \text{ with } i = 1, 2, \ldots, N \text{ and } \sum_i \alpha_i y_i = 0. \end{aligned} \tag{10}$$

The Linear SVM algorithm is implemented using the `LinearSVC` function of the `scikit-learn` library in Python. Default parameters are used, tuning the hyperparameter C using a 5-fold cross validated grid-search over the values $C = \{0.01, 0.1, 1, 10, 100\}$.

Logistic Regression: The second classifier that is used is the Logistic Regression (LR), also known as the Multinomial Logistic Regression in a multiclass setting. The logistic regression is commonly used to model a binary/ordinal dependent variable based on one or more independent variables (predictors), assuming a linear relation between the log-odds and the predictors. In the logistic regression, the logistic function is used to convert log-odds to probabilties, although there also exists analogous models that use alternative functions (e.g. sigmoid function in the probit model). Technically, the logistic regression is not a classifier, as it only models the probability of a certain output class based on the input features. However, using the class probabilities, one can easily assign observations to classes.

Mathematically, under a multinomial logistic regression model, the probability that observation $\boldsymbol{x}$ belongs to class i is defined as shown in Equation 11. Here, we follow the notation that an example belonging to class i is denoted as $y^{(i)} = 1$ and $y^{(i)} = 0$ otherwise. It follows that:

$$P\left(y^{(i)} = 1 \middle| \boldsymbol{x}, \boldsymbol{w}\right) = \frac{\exp\left(\boldsymbol{w}^{(i)^T}\boldsymbol{x}\right)}{\sum_{j=1}^{m}\exp\left(\boldsymbol{w}^{(j)^T}\boldsymbol{x}\right)}, \tag{11}$$

for $i \in \{1, \ldots, m\}$, where $\boldsymbol{w}^{(i)}$ denotes the weight vector corresponding to class i and the superscript ^{T} implies a vector/matrix transpose.

Because of the normalization condition on the class probabilities, the weight vector for one of the classes does not need to be estimated. Therefore, without loss of generality the last weight vector can be set equal to zero $\boldsymbol{w}^{(m)} = \mathbf{0}$. The remaining model parameters are determined using maximum likelihood estimation, maximizing $\prod_{j=1}^{n} P\left(\boldsymbol{y}_j | \boldsymbol{x_j}, \boldsymbol{w}\right)$, where $\boldsymbol{w}$ denotes the concatenated vector of parameters to be learned. This corresponds to maximizing the log-likelihood

function:

$$\begin{aligned}\mathcal{L}(\boldsymbol{w}) &= \sum_{j=1}^{n} \log P\left(\boldsymbol{y}_j | \boldsymbol{x_j}, \boldsymbol{w}\right) \\ &= \sum_{j=1}^{n} \left[\sum_{i=1}^{m} y_j^{(i)} \boldsymbol{w}^{(i)^T} \boldsymbol{x}_j - \log \sum_{i=1}^{m} \exp\left(\boldsymbol{w}^{(i)^T} \boldsymbol{x}_j\right) \right].\end{aligned} \tag{12}$$

In practice, the Logistic Regression is often used in combination with some form or regularization, in order to prevent overfitting of the classification algorithm. This thesis uses l_2 regularization, which adds a penalty term $\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}$ to the log-likelihood function in Equation 12. In order to tune the regularization, a constant C is added to the non-regularization part of the log-likelihood function, resulting in the following regularized log-likelihood function:

$$\mathcal{L}(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C \sum_{j=1}^{n} \left[\sum_{i=1}^{m} y_j^{(i)} \boldsymbol{w}^{(i)^T} \boldsymbol{x}_j - \log \sum_{i=1}^{m} \exp\left(\boldsymbol{w}^{(i)^T} \boldsymbol{x}_j\right) \right], \tag{13}$$

which is minimized using the SAGA solver (Defazio, Bach & Lacoste-Julien, 2014) implemented in the `scikit-learn` library in Python. It follows from Equation 13 that setting a higher value of C results in less regularization, as the part $\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}$ becomes relatively smaller compared to the total value of the log-likelihood function. The hyperparameter C is chosen in a similar way as with the support vector machine, using a 5-fold cross validated grid-search over the values $C = \{0.01, 0.1, 1, 10, 100\}$.

4.2 Visual features

Besides classifying our documents using purely textual features, we also investigate classification using only visual features, before looking into a classification method that combines both types of features. This section further explains how the visual classification is performed.

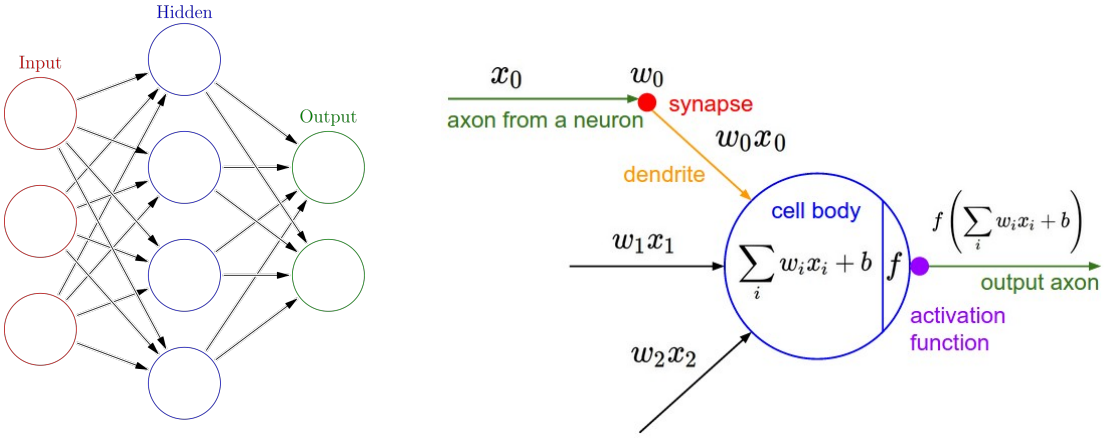
From the literature review it follows that current state-of-the-art visual classification performance is obtained by using (deep) convolutional neural networks (CNN). Therefore, this thesis focuses on using a CNN to perform visual classification. For doing so, the methodology for this thesis loosely follows the methodologies of Afzal et al. (2017) and Das et al. (2018).

The main idea that this thesis follows when performing visual classification is: extracting visual features by using a pre-trained VGG16 model (Simonyan & Zisserman, 2014) and consequently using the extracted visual features to train a shallow neural network model, in order to perform (visual) classification. The subsequent parts cover: the general architecture of a neural network, some specifics for convolutional neural networks, the concepts of backpropagation and stochastic gradient descent, and the model design used in this thesis.

4.2.1 Neural Network architecture

In Figure 6a, a schematic overview of a neural network architecture can be seen. The basic idea of a neural network is to create a mapping between an input layer and an output layer, through a series of in between (hidden) layers. The input layer takes in raw data, in this case

the pixel values of input images, and the output layer returns the wanted output based on the input features, in this case the classification score for each category. Consequently, the input document is classified into the category that gets the highest score. The small circles in each layer in Figure 6a are so-called 'neurons', a central concept within the framework of neural networks.



(a) Network design

(b) Neuron

Figure 6: Schematic overview of Neural Network concepts

To explain the concept of a neuron, in general the analogy with a neuron within the human brain is used. In our brains, neurons receive input through dendrites, which are connected to other neurons. If enough input is received, a neuron will become active, resulting in an output signal, which will be the next neuron's input. A schematic overview of a neuron's mathematical equivalent, which forms the basis of neural networks, is shown in Figure 6b.

In Figure 6b, three inputs x_i can be seen, each with their corresponding weights w_i. The weights w_i reflect the degree of importance of the given connection in the neural network. Within the neuron in Figure 6b, a weighted sum z is taken over the inputs, including a bias term b. Afterwards, the sum is taken as input for the (non-linear) activation function f, which forms the output of the neuron shown. In general, a single activation function is a relatively simple mathematical transformation. However, when stacking several layers of non-linear functions, the resulting network can capture complex, highly non-linear patterns, resulting in often very good classification performance.

In matrix notation, the mathematical relation between different layers can be described as:

$$\begin{aligned} \boldsymbol{z}^l &= \boldsymbol{w}^{l^T} \boldsymbol{a}^{l-1} + \boldsymbol{b}^l, \\ \boldsymbol{a}^l &= f(\boldsymbol{z}^l), \end{aligned} \tag{14}$$

where $\boldsymbol{a}^l$ is the activation vector of layer l and $\boldsymbol{z}^l$ is the weighted input to the neurons in layer l. The latter is linearly dependent on the activation vector $\boldsymbol{a}^{l-1}$ of the previous layer, the corresponding weights $\boldsymbol{w}^l$ and the corresponding bias vector $\boldsymbol{b}^l$. Moreover, the superscript ^{T} again implies a vector/matrix transpose and f denotes the (non-linear) activation function that

is applied element wise to the weighted sums. Going forward, we will use matrix notation, but it is good to note that the jth element of $\boldsymbol{z}^l$ can be calculated as follows: $z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$, with k in this case being the number of neurons in the $(l-1)$th layer.

The exact type of activation function differs for different types of neurons. A typical example of a type of neuron is a binary logistic unit, also known as sigmoid neuron, for which the activation function is given by: $f(z) = \frac{1}{1+e^{-z}}$. Other commonly used neurons types are the perceptron, tanh and rectified linear unit (ReLU). This thesis uses the ReLU (Nair & Hinton, 2010) as activation function for the hidden layers, as it is considered to be a good default choice for the hidden layers by Goodfellow, Bengio & Courville (2016). The reasons Goodfellow et al. give for this are: the ReLU is easily optimized, has a large and consistent derivative at each point where the ReLU is active and has a second derivative that is zero in almost all situations. Formally, the ReLU is defined as:

$$f(z) = \max(0, z). \tag{15}$$

The only thing to note when using the ReLU is that Equation 15 is not differentiable at $z = 0$. However, in practice software implementations usually return either the left or the right derivative, which can be heuristically justified by the observation that the gradient-based optimization is subject to numerical (rounding) errors anyway (Goodfellow et al., 2016).

The activation function used in the output layer is the softmax function, as we would like to represent a probability distribution over the number of classes in our classification problem. Intuitively, the softmax function can be seen as a generalization of the sigmoid function. Mathematically, it is defined as:

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{n} \exp(z_j)}. \tag{16}$$

Attractive properties of the softmax function are that exponentiation results in strictly positive values, while the denominator ensures that the output values over all classes together sum to one.

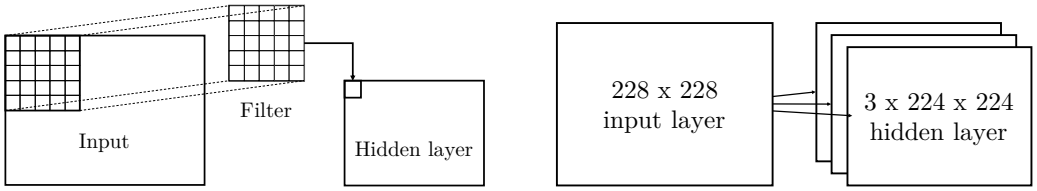
4.2.2 Convolutional Neural Networks

The hidden layers in a neural network are formed by neurons, that apply simple mathematical operations on the weighted inputs they receive, running the result of the operation through an activation function onto the next hidden layer. A special mathematical operation that is often used when processing data that has a grid-like structure (such as the pixels in images), is the convolution. Neural networks that apply this operation are therefore known as convolutional networks or convolutional neural networks (CNNs) (LeCun, Boser, Denker, Henderson, Howard, Hubbard & Jackel, 1989). This subsection explains the theory of a CNN based on three concepts: local receptive fields, weight sharing and pooling, following the work of Nielsen (2015).

When explaining the concept of a CNN, it helps to think in two (or three) dimensions, corresponding to for example an image. A local receptive field can be thought of as a little

window, that slides over the input layer (image), thereby forming a connection between the input layer (e.g. the pixels of an image) and the hidden layer. To be more precise, each neuron in the hidden layer is connected to a small region of the input layer, for example a region of 5×5 pixels. This concept can be seen in Figure 7a. The complete hidden layer is formed by sliding the window over the complete input layer, usually moving one pixel at the time, although different stride lengths can be used.

The value of the neuron in the hidden layer can then be calculated by taking the values of the input layer and multiplying them with the 5×5 weights corresponding to the local receptive field (window), summing the result, adding it's bias and applying the nonlinear activation function. Important to note is that the weights corresponding to the window are *shared*, in contrast to the situation described before, where each neuron had it's own weights. Because of the shared weights, one can think of the sliding window as detecting the same feature in different locations in the image, for example detecting a vertical edge throughout the complete picture. The combination of the local receptive field with it's shared weights and bias is often referred to as filter or kernel. Moreover, the resulting hidden layer that is obtained by applying a specific filter over the complete input layer is referred to as feature map. In order to detect multiple features, a convolution layer usually consists of multiple feature maps, as is shown in Figure 7b.



(a) Local receptive field (b) Multiple feature maps

Figure 7: Schematic overview of CNN concepts

The third concept that is usually present in CNNs is the concept of a pooling layer. A pooling layer is often used directly after one or more convolutional layers. Essentially, a pooling layer simplifies the output of the convolutional layer, by repeatedly summarizing small input regions, in a similar fashion as convolutional layers do. An example of a pooling layer is a layer where each neuron takes the maximum value of 2×2 neurons in the previous layer. This is known as max-pooling, but other pooling techniques such as average-pooling or L2 pooling might also be used. Intuitively, pooling can be thought of as asking the network whether a given feature is present somewhere in a region of the image, but throwing away the exact positional information. In the context of image classification, this would translate to checking whether a certain logo is present somewhere in the header of the image, although the exact position might differ from document to document. An added benefit of applying a pooling layer is that it greatly reduces the dimensionality of the neural network, thereby increasing the statistical efficiency of the network (i.e. reducing the computational cost of training the network). An example of a convolutional layer followed by a pooling layer is shown in Figure 8.

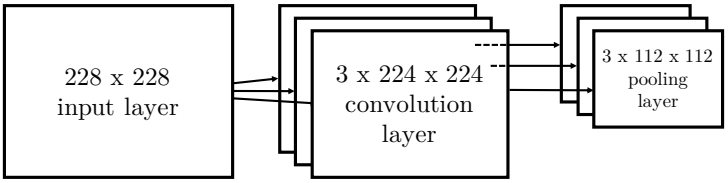


Figure 8: Schematic overview convolutional + pooling layer,
based on figure in Nielsen (2015)

4.2.3 Backpropagation

In the previous section, it was shown how the input features $x_1^0, \ldots, x_k^0$ form the neurons in the first (input) layer $\boldsymbol{l}^0$. The next layer is then formed by taking the summed weighted values of these neuron and transforming them by means of an activation function, to form the second layer in the network, being the first hidden layer $\boldsymbol{l}^1$. Consequently, the same steps are repeatedly applied for all N hidden layers and for the output layer $\boldsymbol{l}^{N+1}$. The result is a mapping of input features to a number of output neurons, that correspond with the number of classes, in case of a classification problem.

When training a neural network, the goal is to learn the weights that determine the mapping between the different layers in the network. This is done by minimizing a loss function, that has to be chosen by the researcher. Although finding the optimum of the loss function may be done analytically in a small neural network setting, this is usually not possible when the network size increases due to the large number of variables. In order to minimize the loss function and optimize the learned weights, neural networks nowadays usually use a technique called backpropagation at training time (Rumelhart, Hinton, Williams & others, 1988).

The goal of backpropagation is to compute the partial derivatives $\frac{\partial \mathcal{L}}{\partial w_{jk}^l}$ and $\frac{\partial \mathcal{L}}{\partial b_j^l}$ of the loss function $\mathcal{L}$ with respect to any weight w_{jk}^l or bias b_j^l in the network, and consequently use these partial derivatives to minimize the overall loss function using Stochastic Gradient Descent (SGD) (Nielsen, 2015).

Before we derive the partial derivatives, we first note that $\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^l} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{a}^l} \frac{\partial \boldsymbol{a}^l}{\partial \boldsymbol{z}^l} \frac{\partial \boldsymbol{z}^l}{\partial \boldsymbol{b}^l}$ and $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^l} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{a}^l} \frac{\partial \boldsymbol{a}^l}{\partial \boldsymbol{z}^l} \frac{\partial \boldsymbol{z}^l}{\partial \boldsymbol{W}^l}$ share a common component: $\frac{\partial \mathcal{L}}{\partial \boldsymbol{a}^l} \frac{\partial \boldsymbol{a}^l}{\partial \boldsymbol{z}^l}$. Therefore, we define the local error δ_j^l of neuron j in layer l as:

$$\delta_j^l \equiv \frac{\partial \mathcal{L}}{\partial z_j^l}. \tag{17}$$

Next, we define an equation for the error δ_j^{N+1} in the output layer $\boldsymbol{l}^{N+1}$. Moreover, we define a relation between the errors in different layers, which we can use to propagate the error *backward* through the network. Consequently, by combining these two equations, we are able to easily calculate the error for each layer, which is the strength of the backpropagation algorithm.

First, define the influence of the error δ_j^{N+1} of neuron j in the output layer on the loss function as:

$$\delta_j^{N+1} = \frac{\partial \mathcal{L}}{\partial a_j^{N+1}} f'(z_j^{N+1}). \tag{18}$$

Looking at Equation 18, two terms can be seen. The first term, $\frac{\partial \mathcal{L}}{\partial a_j^{N+1}}$, measures the degree of change in the loss function, as function of the j-th output activation. From the chain rule follows the second term, $f'(z_j^l)$, the derivative of the activation function used in the output layer, evaluated in z_j^{L+1}. As equation 17 is defined component wise, it can rewritten in matrix-based form to ease further derivation of the backpropagation formula's:

$$\boldsymbol{\delta}^{N+1} = \nabla_a \mathcal{L} \odot f'(\boldsymbol{z}^{N+1}), \tag{19}$$

with $\nabla_a \mathcal{L}$ being a k-dimensional vector containing the partial derivatives $\frac{\partial \mathcal{L}}{\partial a_j^{N+1}}$ for all k neurons in the output layer, $f'(\boldsymbol{z}^l)$ containing in a similar fashion the partial derivatives $f'(z_j^{N+1})$ for $j \in 1, \ldots, m$ and $\odot$ being the Hadamard product.

As a second step, define an equation for the error δ^l in terms of δ^{l+1}, the error in the next layer:

$$\boldsymbol{\delta}^l = ((\boldsymbol{w}^{l+1})^T \boldsymbol{\delta}^{l+1}) \odot f'(\boldsymbol{z}^l), \tag{20}$$

with w^{l+1} being the weight matrix for the $(l+1)$th layer, containing the weights between the the lth and the $(l+1)$th layer. Intuitively, Equation 20 shows how the error at the $(l+1)$th layer passes through the network to the previous layer, via the transposed weight matrix. By taking the Hadamard product with $f'(\boldsymbol{z}^l)$, the error subsequently moves backwards through the activation function in layer l, which gives the result in Equation 20.
Using Equation 14, Equation 19 and Equation 20, it is possible to calculate the partial derivatives $\frac{\partial \mathcal{L}}{\partial w_{jk}^l}$ and $\frac{\partial \mathcal{L}}{\partial b_j^l}$ as:

$$\frac{\partial \mathcal{L}}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \qquad \text{and} \qquad \frac{\partial \mathcal{L}}{\partial b_j^l} = \delta_j^l. \tag{21}$$

4.2.4 Stochastic Gradient Descent

Using the partial derivatives calculated by means of backpropagation, the loss function can be minimized using iterative optimization algorithm such as Gradient Descent. The idea of Gradient Descent is to calculate the gradient of $\mathcal{L}(\theta)$ for the current value of θ, and then take a small step in the direction of the negative gradient. Mathematically, this can be denoted as:

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta \mathcal{L}(\theta), \tag{22}$$

with θ^{new} being the updated parameter values, $\nabla_\theta \mathcal{L}(\theta)$ being the gradient of $\mathcal{L}(\theta)$ and α being the step size or learning rate. Because the loss function is in general not convex, it is not guaranteed that the gradient descent algorithm converges to a global solution. It might also be the case that the found optimal solution in fact is only a local optimum, or even that the algorithm does not converge at all within the specified number of iterations.

However, next to these problems, calculating the gradient itself is computationally expensive when the number of training observations increases. Therefore, one can use Stochastic Gradient Descent (SGD) in order to reduce the computation time Bottou (2010). The idea of SGD is:

instead of calculating the gradient $\nabla_\theta \mathcal{L}(\theta)$ completely, estimate the gradient in each iteration using a random sample of m' observations $\mathbb{B} = \left\{ \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m')} \right\} \in \boldsymbol{X}$, also known as a mini-batch. Provided that m' is large enough, the calculated average gradient over the mini-batch is expected to be roughly equal to the average gradient over the whole sample, being:

$$\frac{\sum_{j=1}^{m} \nabla_{\theta, x_j} \mathcal{L}(\theta)}{m'} \approx \frac{\sum_X \nabla_{\theta, x} \mathcal{L}(\theta)}{n} = \nabla_\theta \mathcal{L}(\theta), \tag{23}$$

where the last part holds provided that the chosen loss function is such that $\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_i(\theta)$. In practice, choosing a large value for m' might result in a slower convergence speed, due to less added variance in the optimization problem, while choosing a small m' might result in noisy gradient estimates (Li, Zhang, Chen & Smola, 2014). Typically, the mini-batch size m' is chosen to be relatively small, ranging from one to a few hundred observations (Goodfellow et al., 2016).

Next to plain (vanilla) SGD, a multitude of related optimization techniques have been proposed in literature that automatically adapt the learning rates of model parameters, such as AdaGrad (Duchi, Hazan & Singer, 2011), RMSProp (Tieleman & Hinton, 2012) and Adam (Kingma & Ba, 2014). This thesis uses the Adam algorithm, the most recent of these three, although no major difference was observed when using RMSprop.

4.2.5 Model design

The model design for the document classification based on the visual features is inspired by the methodology of Das et al. (2018). The basis is formed by the VGG16 model (Simonyan & Zisserman, 2014) trained on the popular ImageNet data set (Deng et al., 2009). A schematic overview of this model is shown in Figure 9.

From the pretrained VGG16 model the top layers are excluded, keeping the rest of the layers as feature extraction layers. This means that the first five blocks (18 layers in total) of two (three respectively) convolutional layers followed by a pooling layer are used with pretrained weights. Based on the output of the last pretrained layer a new set of top layers is trained, in order to finetune the neural network for the specific data set at hand. To be able to use the data set with the VGG16 model, as a preprocessing step the input document are resized to be 224 by 224 pixels and standardized.

Because of the computational complexity, a limited set of design options is considered for the shallow top network. For the Rabobank data set, the dimensionality of the fully connected top layer(s) is initially varied to be 1024, 2048 or 4096 wide, while also varying the number of fully connected layers between one and two. In total, this gives six different models that are tried in a cross-validation setting. The dimensionality of the top layer is explicitly chosen to be smaller than the top layers in the original VGG16 model, because of the relatively limited size of the Rabobank data set. For the fully connected layers the ReLu activation function is used. Moreover, the Adam optimizer is used with a learning rate of 2e$-$4, using categorical cross-entropy as loss measure, optimizing the accuracy in the validation fold. As regularization measure we use a dropout of 0.5 between the fully connected layers.

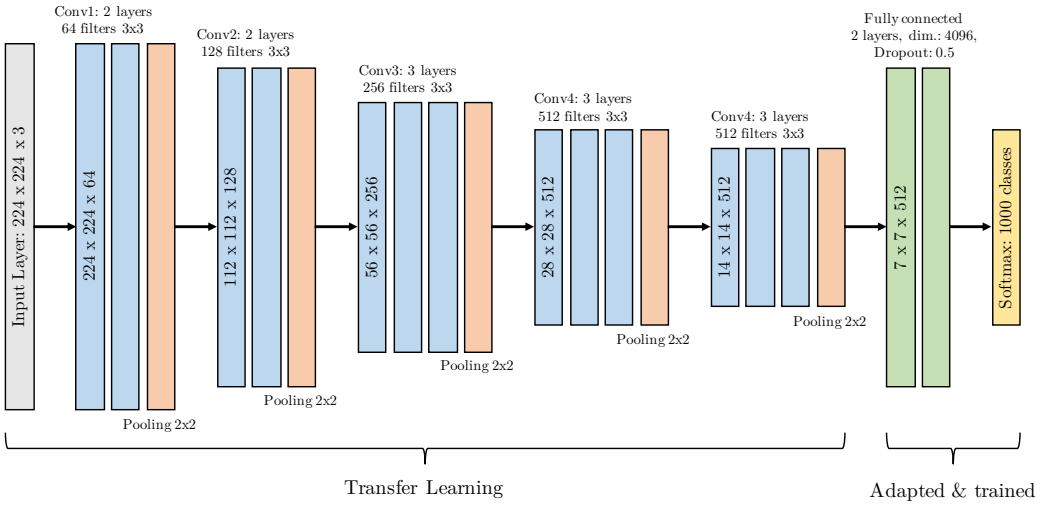


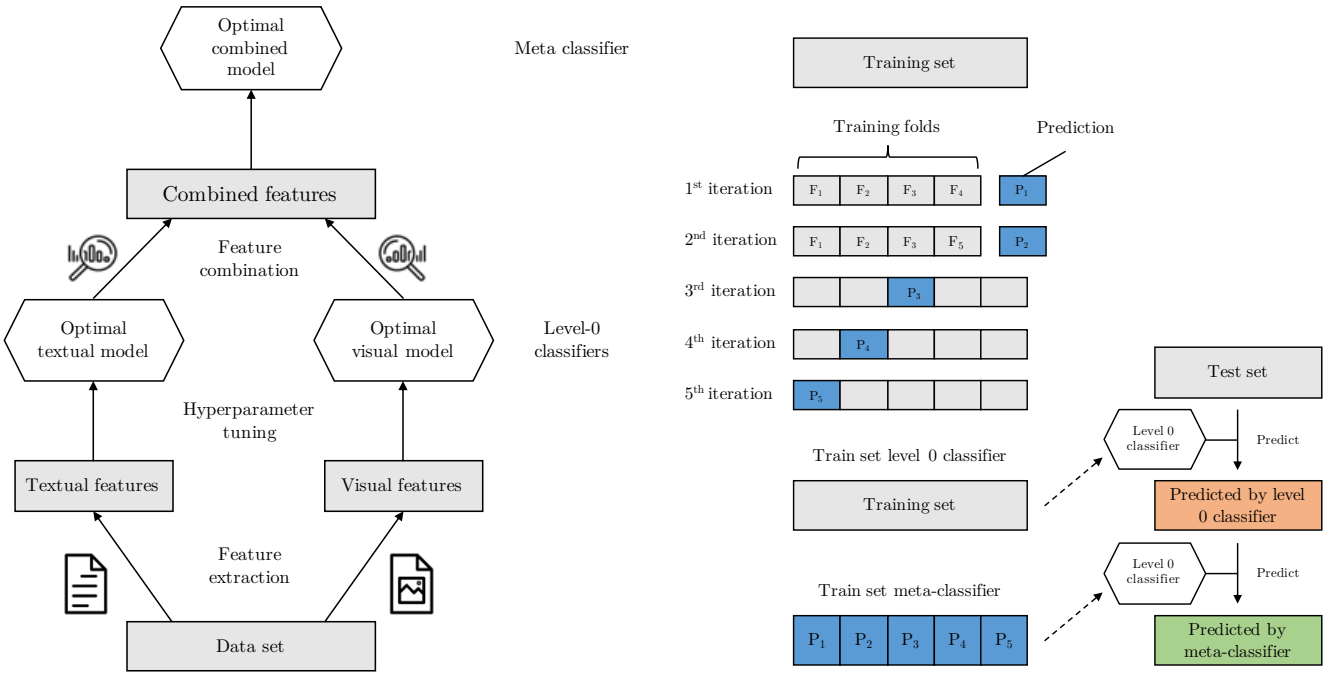
Figure 9: Schematic overview of model architecture based on VGG16 model, based on figure in Das et al. (2018, p. 3)

For the RVL-CDIP data set, a slightly different model design strategy is considered. With respect to the design of the CNN, we adhere to the VGG16 model, with two top layers consisting of 4096 fully connected neurons. Instead, we vary the number of layers that are retrained. Initially, the same approach is used as with the Rabobank, extracting visual features using the convolutional blocks of the VGG16 network and (re)training the two fully connected top layers. Afterwards, another model is estimated, this time extracting visual features using only the first four convolutional blocks (the first 14 layers), (re)training both the last convolutional block and the two fully connected top layers. By doing so, more parameters are *trainable*, resulting in a more flexible model, that should theoretically be able to yield better classification performance.

The reasons for using this different model design strategy are threefold: first of all, because the RVL-CDIP data set is considerably larger than the Rabobank data set, it is not needed to change the dimensionality of the top layers. Second of all, adhering to the model architecture of the VGG16 model enables better comparison with previous results in literature. Last of all, by varying the number of trainable layers, the computational complexity of the model is also varied. This gives insight in the trade off between statistical performance (classification accuracy) and the computational performance (amount of time needed for training/scoring).

4.3 Combination of features

After classifying our data sets based on purely textual and purely visual features, it is of interest to see whether combining both types of features can give more classification power. In order to combine our features, we use decision-level fusion. This means that the features are not directly combined by concatenating them, but that the both types of features are used through the previously made classification models, with their respective optimal hyperparameters. A schematic overview of this approach is shown in Figure 10a. These classifiers, the logistic



(a) Model stacking

(b) Training meta-classifiers

Figure 10: Schematic overview of feature combination

regression for the textual features and the CNN for the visual features, are referred to as the level-0 classifiers. In order to combine the predicted class probabilities of the level-0 classifiers, a meta-classifier is used. We consider two deterministic methods and three probabilistic models as meta-classifiers. The next section describes each of the combination methods, followed by a short discussion on how the meta-classifiers are fitted.

Meta-classifiers: Five different methods are considered to combine the predicted class probabilities obtained from the level-0 classifiers, two deterministic methods and three probabilistic methods. The baseline is formed by two simple deterministic methods: averaging and multiplying the predicted class probabilities. These methods are used because of their simplicity, resulting in both fast and understandable models, that do not need additional training. Next to these deterministic methods, three probabilistic models are used as meta-classifiers. For these, we choose the same models as used before; a logistic regression and a linear support vector machine. Next to that, we add a non-linear classifier, being a support vector machine model with the radial basis function kernel. This non-linear model is added in order to see whether this adds additional classification performance over the two other linear models.

Training the meta-classifier: The stacking of models also has implications for the way in which the meta-classifier is trained. The meta-classifier takes as input the class probabilities predicted by the level-0 classifiers. Therefore, we need (unbiased) predictions for the train set, made for observations that have not been used in the training of the level-0 classifiers. In

order to generate these unbiased predictions, we essentially use a variation on the k-fold cross validation algorithm, as shown in Figure 10b.

First, we split our training data in k folds ($k = 5$ in this thesis), denoted as $F_1, \dots F_5$ in Figure 10b. Next, we fit a level-0 classifier on $k-1$ folds and predict the k^{th} fold, denoted as P_1. We repeat this step in order to generate predictions $P_1, \dots, P_5$ for each of the k folds, as shown in the upper part of Figure 10b. Afterwards, the predicted folds are combined and used to generate training data for the meta-classifier. Moreover, the level-0 classifier are retrained on the entire training set. This is done for each of the level-0 classifiers, in our case for both the textual and the visual features. When evaluating the model, a sample document from the test set is first predicted by the level-0 classifiers. The combination of the level-0 predictions is then used by the meta-classifier to form the final prediction (left part of Figure 10b). In order to tune the hyperparameters of the meta-classifier (if needed), we use 5-fold cross-validation using the combined predictions that form the training data for the meta-classifier.

4.4 Model evaluation

In order to evaluate the model performance, the primary performance measure will be the *accuracy* of the predictions. However, it is known that accuracy tends to favor models that primarily predict the larger classes. Therefore, accuracy is a less favourable evaluation measure for data sets that are highly imbalanced. In order to further evaluate the model performance with respect to the minority classes, this thesis also looks at the macro-averaged F_1-measure (Van Rijsbergen, 1986).

The macro-averaged F_1-measure can be calculated by taking the (unweighted) average of the $F_{1,c}$-measure's for each of the different classes c. In general, the F_1-measure is calculated as follows:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}, \tag{24}$$

with precision defined as the ratio between the number of correct predictions and the total number of predictions for a specific class, and recall defined as the ratio between the number of correct predictions and the total number of observations in a specific class. Mathematically, that is:

$$\begin{aligned} Precision &= \frac{\text{true positives}}{\text{true positives} + \text{false positives}} \\ Recall &= \frac{\text{true positives}}{\text{true positives} + \text{false negatives}} \end{aligned} \tag{25}$$

5 Results

This section presents the results of this thesis. First, the results obtained for the Rabobank data set are discussed, followed by the results obtained for the RVL-CDIP data set.

5.1 Rabobank

In this section, we follow the structure that has also been used in the methodology section, starting with classification based on textual features, followed by classification based on visual features and concluding with classification based on the combination of both feature types. First, we start with some preliminary results for the textual feature extraction.

Textual feature extraction

As a preliminary analysis, we investigate to what extent tf-idf is able to extract textual features from the documents that seem intuitively correct. Therefore, we calculate the tf-idf scores for the complete data set, average them over the separate classes and evaluate whether the top 10 terms per class can be considered reasonable (for that class). As there are 17 classes, with similar patterns observed for different classes, we only show the results for four classes in Figure 11.

In general it is observed that tf-idf is reasonably able to extract relevant features for the Rabobank data set, as the terms match with what is expected based on the class definition. Moreover, the effects of stemming and adding bigrams can clearly be seen. Next to that, some interesting things are observed in Figure 11. First of all, it is seen that (part of) the class name sometimes comes up as top tf-idf term, for example in Figure 11b. Next to that, it can also be seen that the same term sometimes shows as top tf-idf term for another class, as we observe the term *borg* not only in the category 'Borgtocht', but also for the category 'Zekerhedenregeling' in Figure 11a.

Besides the terms that are (closely related to) class names, we also observe terms that, at first sight, appear to be quite general. An example of this is shown in Figure 11c, where the top terms are *any*, *or* and *the*. Interestingly, these terms are not only in English, but also words that would normally be filtered when using an English stopword list. However, when taking into account the class for which these terms are found (Correspondence), it becomes clear that these terms are probably found in the English disclaimer that is often included at the bottom of sent emails. Further evaluation shows that the disclaimer indeed includes seven of the ten terms shown in Figure 11c. Next to that, it can be seen that the average tf-idf score for this class is much lower than for the other classes shown, which could be an indication that this class is more difficult to classify and why relatively general terms come up as top tf-idf terms for this class.

Last of all, we observe in Figure 11d that sometimes class names are not shown as top tf-idf terms, although these class names are often found in documents of these classes. An

(a) Zekerhedenregeling (b) Borgtocht (c) Correspondence (d) KvK extract

Figure 11: Top tf-idf terms for different classes

example is shown for the class in Figure 11d (Invoices). We had expected that the term *factuur* would be shown, as a visual inspection showed that this term is included in virtually every invoice document. However, this term is not found in the top 10 tf-idf terms, and upon further inspection also not found in the top 25 terms. Our hypothesis is that this is related to the downweighting of frequent terms in the tf-idf algorithm. As *factuur* is almost always shown on documents within this class, combined with the fact that the invoice category is relatively large compared to the entire document corpus, the result is that the word is downweighted and therefore not shown in the top tf-idf terms.

5.1.1 Textual feature classification

Next, we turn towards the classification results based on the textual features. As described in Section 4.1, a 5-fold cross-validated gridsearch was used over a set of hyperparameters in order to find the classification model that yields the best result. Afterwards, this best model was retrained using the complete training set and scored on the previously unseen test set.

From the hyperparameter search, several things can be seen immediately. First of all, it is seen that models that apply l_2 normalization strictly outperform models that use l_1 normalization. The latter models obtain accuracy figure ranging from 60% to 90% (averaged of the 5 folds), while the former models consistently score above 90% average accuracy. After correcting for normalization, we find that models using the logarithmically scaled term frequency outperform models that use the raw count as term frequency in all but a few cases. Next to that, it is observed that linear SVM models mostly outperform logistic regression models, except for when the logistic regression model is tuned with a larger regularization constant C (implying less strict regularization). Interestingly, the support vector machine algorithm is relatively robust for different values of C, whereas the classification performance of the logistic regression decreases for smaller values of C to figures below 90% accuracy. This suggests that overfitting is not much of a concern in the current application.

For the other hyperparameters, the effects are less clear. It is seen that different combinations of hyperparameters yield slightly different classification performances, but both in terms of absolute performance and in terms of variation between different folds, models score similarly.

Therefore, it appears that choosing the classification method and tuning its hyperparameters is more important than the tuning the hyperparameter of the feature extraction method.

Table 1: Results textual hyperparameter gridsearch

Classifier	**Optimal Hyperparameters**						**Performance CV**[1]		**Performance OOS**[2]	
	N-grams	Min. df	Max. df	Max. Features	Norm.	Term Frequency	Accuracy	F1 (macro)	Accuracy	F1 (macro)
LinearSVM (C=1)	(1,2)	2	*None*	*None*	l_2	$\log(1 + f_{t,d})$	0.9597	0.9317	0.9622	0.9333
Logistic Regression (C=10)	(1,2)	5	0.5	*None*	l_2	$\log(1 + f_{t,d})$	0.9568	0.9271	0.9606	0.9177

[1] Performance CV denotes the average classification performance obtained in the 5-fold cross validation.

[2] Performance OOS denotes the classification performance when scored on the quasi out-of-sample test set.

In the end, the best classification performance using only textual features was obtained with a linear SVM classifier with regularization constant $C = 1$, using both unigrams and bigrams, combined with a minimum document frequency (for terms to be included) of two, and no maximum number of features. This model, together with the best scoring logistic regression model is shown in Table 1. The linear SVM model was able to classify 95.97% of the documents correctly, averaged over the five different folds. Scoring this model on the test set, which was held out in the hyperparameter tuning, led to a marginally higher classification performance of 96.22%. From the classification report in Table 10 (Appendix A) it can be seen that the model performs quite robust over different classes, obtaining a macro averaged F1-score of 0.933. The best scoring logistic regression model did not perform much worse, and was found to score only 0.3% less accurate (in absolute terms) averaged over the five different folds. When scoring on the held out test set the differences was even smaller, with the logistic regression model obtaining 96.06% accuracy.

5.1.2 Segmented textual feature classification

One of the primary disadvantages of tf-idf is that the order of words in a body of text is ignored. This not only means that the context in which a word is placed is ignored, but also that it is not taken into account in which section a word is used. Whereas losing contextual information is often a problem when doing sentiment analysis, this information is usually less important for document classification. However, 'sectional' information often *is* important when classifying documents, as headers or titles sometimes contain more information than the body of a text. This is for example the case for invoices or contracts, that often mention their document type somewhere on top of the first page.

Therefore, we have adapted the tf-idf algorithm to run on specific segments of a document, in order to capture the sectional information of the textual features in a document. As different document types can potentially have different document sizes, the segments are defined to split a document in $n \times m$ equal rectangular parts, for the x and y axis respectively. This implies that a segmentation of $(x, y) = (2, 2)$ refers to splitting a document in four equal quarters, while a segmentation of $(x, y) = (4, 1)$ refers to splitting a document only on the x-axis, and $(x, y) = (1, 4)$ refers to splitting a document in a similar manner on only the y-axis.

Running the textual feature extraction in a segmented fashion introduces an additional

hyperparameter, being the number of segments to use. As the documents in our set are in general portrait oriented pages of size A4, we choose to limit the number of segments to a maximum of 20 segments, dividing the x-axis in four parts and the y-axis in five parts. Next to that, we try somewhat less granular segments with $(x, y) = (3, 4)$ and $(2, 3)$ respectively. Moreover, we try segmenting the document over only the x or only the y axis, as this potentially helps classifying document categories that mainly contain tables, such as the *Checklist* category. Together with the $(x, y) = (2, 2)$ segmentation, this forms a total of six different segmentations that are initially tried. The other hyperparameter settings are taken from the (optimal) results of the gridsearch ran previously (see Table 1). Therefore, we use a LinearSVM as classification algorithm including unigrams and bigrams, with the other hyperparameters following from Table 1.

The results for the 5-fold cross validated segmented textual classification are shown in Table 2. It can be seen that the accuracy scores for the different models are very close to eachother, with the worst performing model scoring an accuracy that is only 0.29% lower in absolute terms than the best performing model. Next to that, it is seen that for the 5-fold CV only the best scoring segmented model, with segments $(6, 1)$ and 95.98% accuracy, is able to outperform the best performing unsegmented model, which scored 95.97% accuracy. However, because of the segmentation, the feature extraction process is much more computationally expensive, resulting in train and score times that scale with the number of segments used.

Table 2: Results initial hyperparameter gridsearch segmented textual models

Classification model		**Computational cost**[1]		**Performance CV**[2]	
Algoritm	Segments (x,y)	Train time (min.)	Score time (min.)	Accuracy	F1-macro
LinearSVM (C=1)	(1, 10)	14.32	17.66	0.9577	0.9312
	(2, 2)	14.25	22.41	0.9592	0.9377
	(2, 3)	20.94	25.16	0.9591	0.9367
	(3, 4)	22.49	32.19	0.9569	0.9331
	(4, 5)	36.14	49.64	0.9575	0.9302
	(6, 1)	19.35	26.74	0.9598	0.9373

[1] The train and score times are averaged over the five folds in the cross-validation. Results are obtained on a Intel Xeon E5-2667 quadcore processor @ 3.2GHz. Computation times can be influenced by other running processes and should therefore be only seen as indication of the true computational load.

[2] Performance CV denotes the average classification performance obtained in the 5-fold cross validation.

Interestingly, the best scoring segmented model uses only splits on the x-axis, resulting in documents that are divided in six vertical segments. Based on this finding, another set of segmented models is trained, this time using only a varying number of vertical segments. The results of this additional analysis are shown in Table 3. It is seen that varying the number of vertical segments can yield some extra classification performance, with the best scoring model

using not six but four vertical segments. Doing so results in an overall performance of 96.05% accuracy, with an F1-score of 0.938 (averaged of the five folds). Scoring this model on the held-out test set led to a classification performance that is marginally higher, at 96.34%, being 0.12% higher than the best unsegmented model. However, when further inspecting the classification report in Table 12 (Appendix A), no differences are seen for the category *Checklist*, although this mainly contains tabular documents. For other categories there are small differences between the segmented and unsegmented model, but no clear patterns are observed.

Table 3: Results additional hyperparameter gridsearch segmented textual models

Classification model		**Computational cost**[1]		**Performance CV**[2]	
Algoritm	Segments (x,y)	Train time (min.)	Score time (min.)	Accuracy	F1-macro
LinearSVM (C=1)	(2, 1)	11.56	12.33	0.9596	0.9360
	(4, 1)	13.69	17.29	0.9605	0.9380
	(6, 1)	16.62	22.36	0.9598	0.9373
	(8, 1)	22.08	27.40	0.9600	0.9334
	(10, 1)	26.01	32.82	0.9586	0.9320
	(12, 1)	18.19	21.86	0.9602	0.9345

[1,2] See notes in Table 2

Based on these results, we conclude that it is potentially useful to include structural information in our classification models. Therefore, we continue with classification based on visual features, to see whether only using visual (i.e. structural) information is a feasible method to classify documents.

5.1.3 Visual feature classification

After classifying our Rabobank document set based on purely textual features, the next step is to try and classify the documents using only visual features. As described in Section 4.2, we do so by rescaling the document images to 224 by 224 pixels, extracting features using a pretrained VGG16 model and taking these features as input for a shallow, fully connected neural network.

Table 4 shows the result of our 5-fold cross-validated hyperparameter search in the train data. In total, six different models were fitted, experimenting with three dimensionalities (1024, 2048 and 4096) and either one or two layers. However, Table 4 shows only the results of the four smallest models, as we were unable to fit the largest two models to our data set. This was due to not having enough RAM memory to estimate the models. Although the 2048 dimensionality model has two times the number of parameters compared to the 1024 dimensionality model, no improvement is seen in the classification performance. Based on these results, we do not expect that increasing the dimensionality even further to 4096 would have yielded better classification results for this data set.

From the results of the four other models, it can be seen that the smallest model is found to be the best performing model, both in terms of accuracy and in terms of the macro averaged

F1-score. This model, with one layer consisting of 1024 fully connected neurons followed by a softmax layer, obtained 89.54% accuracy in the cross validation and an macro-averaged F1-score of 0.7977. Nonetheless, it is found that the even the best scoring visual classification model performs considerably worse than the previously seen textual models. Similar patterns can be seen when retraining the optimal visual model on the train set and scoring on the test set. In Table 13 (Appendix A) we see that an overal accuracy of 91.04% is obtained, but that the macro averaged F1-score is only 0.8323. This last figure is mainly cause by relatively lower classification performance in class 2, 4, 16, 17 and 26, all classes with less observations.

Moreover, it can be seen in Table 4 that the visual features based CNN models have a much higher computational cost (primarily at train time). Whereas training a linear SVM based on unsegmented textual features took about 20 seconds, with a logistic regression model taking about a minute longer, we find that training our neural network takes at least 28 minutes. Furthermore, it is seen that the training time scales almost linearly with dimensionality, while adding extra layers also increases the training time considerably. An exception to this appears to be our largest model with two layers of 2048 neurons. However, further inspection of our training logs reveal that the training of this model was prematurely stopped, as the model was 'stuck' in a local minimum. We think this problem could be solved by further tweaking the learning rate, but as we do not find our models improving with larger size, we leave this out of scope.

A last observation that is made, is that the model with two layers of 1024 dimensionality scores almost as good as the (best) visual model with one layer of dimensionality size 1024, looking at the overall accuracy. However, when considering the macro averaged F1 score, it is seen that model performance on the smaller classes is actually much worse. We hypothesize that this is related to having not enough training samples for these specific classes, especially considering the deeper model architecture.

Table 4: Results initial hyperparameter gridsearch visual models

Classification model			**Computational cost**[1]		**Performance CV**[2]	
Dimensionality	Layers	# of parameters	Train time (min.)	Score time (min.)	Accuracy	F1-macro
1024	1	25,708,561	28.71	0.42	0.8954	0.7977
1024	2	26,758,161	42.18	0.64	0.8871	0.6815
2048	1	51,417,105	64.67	0.68	0.8943	0.7963
2048	2	55,613,457	32.85	0.73	0.1915	0.0187

[1] The train and score times are averaged over the five folds in the cross-validation. Results are obtained on a Intel Xeon E5-2667 quadcore processor @ 3.2GHz. Computation times can be influenced by other running processes and should therefore be only seen as indication of the true computational load.

[2] Performance CV denotes the average classification performance obtained in the 5-fold cross validation.

For completeness, we ran another 5-fold cross-validation, testing three additional models with smaller dimensionalities. Based on our previous results, we chose single layer models with dimensionalities of 128, 256 and 512. The results of this second cross-validation can be found

in Table 5. In this table, it can be seen that the performance of the smaller models appears not to be much worse than the larger models' performance, looking at the accuracy figure that range from 86.07% (dim. size 128) to 89.20% (dim. size 512). However, when taking the macro averaged F1-scores into account, it is seen that the loss in accuracy is disproportionately high for the smaller classes. Therefore, it is concluded that the best performing visual classification model is the model with one layer of 1024 fully connected neurons (followed by the softmax layer, which is similar for all tested CNN models).

Table 5: Results additional hyperparameter gridsearch visual model

Classification model			**Computational cost**[1]		**Performance CV**[2]	
Dimensionality	Layers	# of parameters	Train time (min.)	Score time (min.)	Accuracy	F1-macro
128	1	3,213,585	5.53	0.33	0.8607	0.5854
256	1	6,427,153	16.72	0.44	0.8796	0.6564
512	1	12,854,289	33.85	0.63	0.8920	0.7565

[1,2] See notes in Table 4

5.1.4 Combined features classification

Besides classifying our document using only textual or only visual features, we now look at the combination of both features types. As both types of features can capture different document characteristics, it is of interest to see whether the combination of features yield better model performance than obtained so far in this thesis. In order to do so, the optimal models found in Section 5.1.1 and 5.1.3 (the level-0 classifiers) are used to obtain a set of predicted class probabilities for each document. However, instead of using the textual linear SVM model, we use the best found logistic regression model as level-0 classifier. The reason for this lies in the fact that no predicted class probabilities are obtained as output for the linear SVM model, but only the signed distance of observations to the fitted hyperplane. For the visual features, the single layer model with dimension size 1024 is used as level-0 classifier. This set of predicted class probabilities, consisting of m predicted probabilities based on textual features and m predicted probabilities based on visual features per observation, are used as input for the combined classification model (the meta-classifier). Here, m denotes the number of classes in our classification problem, which was 17 for the Rabobank document set.

For the meta-classifier, five different methods are considered, as described in Section 4.3. In Table 6 the results for each of the considered methods are shown, together with the best found hyperparameters (if applicable). Recall that the best unsegmented textual classification model yielded 96.22% OOS classification accuracy, with the best visual classification model scoring 91.04% respectively. However, the best textual classification model used the linear SVM as classification algorithm, but the textual feature input of the combined model comes from the logistic regression classifier, which yielded 96.06% OOS classification accuracy. In Table 6 it is seen that only the three statistical meta-classifiers are able to outperform this last figure. In

other words, only the three statistical meta-classifiers lead to better (combined) classification results than would have been obtained when classifying using only the textual features. From the three statistical meta-classifiers, it can be seen in Table 6 that the SVM with linear kernel scores best on OOS accuracy, with 96.42% of the observations classified correctly. The corresponding classification report can be found in Table 14 in Appendix A.

Although the absolute differences in classification performance compared to the previous textual and visual model are quite small, these results do indicate that it is a good idea to combine features. Table 6 shows that the three statistical models give classification results that are better than would have been obtained by classifying on the inputs separately. Moreover, we find that simply combining the inputs by means of averaging or multiplying is not necessarily a good idea, as the obtained accuracy of 95.18% is lower than the 96.06% accuracy that would have been obtained by only using textual features.

Table 6: Results combined features classification

Meta-classifier	**Performance CV**[1]		**Performance OOS**[2]	
	Accuracy	F1 (macro)	Accuracy	F1 (macro)
Average	*n.a.*		0.9518	0.8868
Multiply	*n.a.*		0.9518	0.9145
Logistic Regression (C=100)	0.9567	0.9262	0.9626	0.9294
Linear SVM (C=1)	0.9584	0.9303	0.9642	0.9280
Rbf SVM (C=100)	0.9576	0.9277	0.9630	0.9274

[1] Performance CV denotes the average classification performance obtained in the 5-fold cross validation. As the Average and Multiply method do not have hyperparameters, no cross-validated performance measures are given for these meta-classifiers.

[2] Performance OOS denotes the classification performance when scored on the quasi out-of-sample test set.

This section concludes the results obtained based on the Rabobank data set. We have seen that the absolute differences between different methods and different models were quite small. Therefore, we continue with a larger data set, that is potentially more difficult to classify. This enables us to further validate the results obtained so far.

5.2 RVL-CDIP

Next we turn to the classification results obtained for the (academic) RVL-CDIP data set (Harley et al., 2015). The RVL-CDIP data set is (more or less) comparable with the Rabobank data set, containing documents for which both images and text is available. Moreover, the number of classes (16) is similar as in the previous data set, although they are balanced (unlike the 17 (included) classes in the Rabobank data set). The classes itself are different, but some categories are found in both data sets, such as "correspondence/email" and "factuur/invoice".

Two important data characteristics that differ between the data sets are the size and the age of the data. The RVL-CDIP data is considerably larger than the Rabobank data set and covers $400,000$ documents, equally divided over 16 classes, with $320,000$ documents as train set, and $40,000$ documents as validation and test set respectively. This difference in number of observations could prove important for the visual classification models, as it is known from literature that (deep) CNNs benefit from additional training data. Moreover, the RVL-CDIP data set is older, with documents primarily dating between 1960 and 2002. This could impact the textual classification performance, as preliminary data analysis showed that the OCR quality for older documents is often less than for newer (digitally created) documents.

We continue our results for the RVL-CDIP data set by first classifying using only textual features, then classifying using only visual features and lastly by using the combination of both textual and visual features.

5.2.1 Textual features classification

When classifying our second data set using only textual features, we use a slightly different approach than for the first data set. Because the process of textual feature extraction is computational heavy, we choose to fix the hyperparameters that are related to this step. Moreover, like in Section 5.1.1, the hyperparameters of the textual feature extraction are less important than the hyperparameters of the classifiers used afterwards. Therefore, we do not expect that our results would change much by varying the feature extraction hyperparameters.

Based on the results obtained from the Rabobank data set, we choose to include unigrams and bigrams, we set the minimum term document frequency on five, the maximum document frequency on 0.5, and the maximum number of features on 2,000,000. This implies that unigrams and bigrams that occur in between five and 160,000 documents are included. In total, this led to a dictionary size of 1,942,832 included terms. In Table 7 the classification results using textual features are shown for the RVL-CDIP data set. We see again that the linear SVM model yields slightly better classification performance than the logistic regression, with 86.46% OOS accuracy versus 86.22% for the logistic regression. Moreover, the same optimal hyperparameter values for the regularization constant are found as with the previous data set. For both classifiers, the classification report when scored on the test set can be found in Appendix B in Table 15 and Table 16, respectively.

Table 7: Results textual models

Classifier	**Textual Hyperparameters**[1]						**Performance CV**[2]		**Performance OOS**[3]	
	N-grams	Min. df	Max. df	Max. Features	Norm.	Term Frequency	Accuracy	F1 (macro)	Accuracy	F1 (macro)
LinearSVM (C=1)	(1,2)	5	0.5	2000000	l_2	$\log(1 + f_{t,d})$	0.8642	0.8655	0.8646	0.8663
Logistic Regression (C=10)	(1,2)	5	0.5	2000000	l_2	$\log(1 + f_{t,d})$	0.8609	0.8624	0.8622	0.8638

[1] The tf-idf hyperparameters were fixed for all models, due to limited computational resources. Moreover, it was found that the hyperparameters of the classifiers were more important than the hyperparameters of the feature extraction.

[2] Performance CV denotes the average classification performance obtained in the 5-fold cross validation, done using the train set of 320,000 observations.

[3] Performance OOS denotes the classification performance when scored on the quasi out-of-sample test set (39,999 observations).

5.2.2 Visual feature classification

After having classified the RVL-CDIP data set using textual features, we turn towards visual classification. From literature, it is known that the current best performing visual classification method for this data set was proposed by Das et al. (2018). First they used intra-domain transfer learning, retraining a full VGG16 deep CNN to obtain an accuracy of 91.11%. Afterwards, they used the obtained weights to stack five region based CNNs to obtain a state-of-the-art accuracy of 92.21%.

Because of computational limitations, we were not able to follow the model architecture of Das et al. (2018). As can be seen in Table 8, training their full model for 20 epochs (the mentioned number in the paper of Das et al. (2018)) would take over two weeks on our setup. Instead we created two smaller CNN models, based on features that are extracted from a VGG16 network that was pretrained on the ImageNet data set. Next to that, we used the model weights as obtained and shared by Das et al. (2018) to create predictions for the validation and test set. Interestingly, we were not able to replicate their results, presumably due to using a different software framework[3].

From Table 8 it can be seen that our larger model scored 88.01% and 87.64% accuracy on the validation and test set respectively, after having trained our model for eight epochs. Moreover, we observe that this is about 4% better in absolute terms than the performance of the smaller model, which takes about half the training time. See also Table 17 (Appendix B).

Table 8: Results visual models

Classification model		**Computational cost**[1]			**Performance Val.**[2]		**Performance OOS**[3]	
Name	Model architecture	# of parameters	Train time	Score time	Accuracy	F1-macro	Accuracy	F1-macro
Small model	2 Fully Connected (FC) 4096	119,611,408	33 min.	39 sec.	0.8349	0.8352	0.8355	0.8351
Large model	3 Conv. $3 \times 3 \times 512$ + MaxPool $2 \times 2 + 2 \times$ FC4096	127,485,472	1 hour	100 sec.	0.8801	0.8801	0.8764	0.8761
Literature (replicated)	Hollistic VGG16 (Das et al., 2018)	134,326,096	*n.a.*	350 sec.	0.8644	0.8637	0.8620	0.8612
Literature	Hollistic VGG16 (Das et al., 2018)	134,326,096	*19 hours*	350 sec.			*0.9111*[4]	

[1] The computational costs are indications, as obtained running on the Google Cloud platform, on a 8 core server, with 52gb of RAM and a Nvidia Tesla K80 GPU. Training times are per epoch. Score time is for 40,000 observations.

[2] Performance Val denotes the classification performance obtained when scoring on the held-out validation set.

[3] Performance OOS denotes the classification performance when scored on the quasi out-of-sample test set.

[4] Accuracy figure as mentioned by Das et al. (2018), we could not replicate this due to computational limitations.

Based on the results of Das et al. (2018) we hypothesize that additional training time combined with retraining not only the last layers, could yield another 3.5% classification per-

[3]We have contacted Das et al. to sort out this issue, and would like to thank them for their co-operation in this matter.

formance. However, this thesis implements another strategy. Therefore, we continue with our large visual model and combine this with our logistic regression based textual model.

5.2.3 Combined feature classification

Table 9 shows the results obtained by combining both types of features through the different meta-classifying methods. We observe that all methods yield considerable improvements in classification performance, with the simple averaging method scoring already 92.75% accuracy. We find this improvement quite remarkable considering that the class probabilities that are used as input for the meta-classifier came from classification models that did not score higher than 87.64% accuracy. This proves that combination of features provides additional information over using only textual or visual input.

Next to that, we (again) find that the more advanced statistical meta-classifiers score better than the (simpler) average and multiply method. In Table 9 we observe the best classification performance for the non-linear radial basis function (rbf) kernel SVM meta-classifier. This model, obtains a state-of-the-art classification accuracy of 93.51%. To our knowledge, this is the best classification accuracy seen for the RVL-CDIP data set to date, bringing an improvement of 1.3% in absolute terms over the computationally heavier stacked CNN approach of Das et al. (2018). For this model, the corresponding classification report can be found in Table 18 in Appendix B.

Table 9: Results combined features classification

Meta-classifier	**Performance CV**[1]		**Performance OOS**[2]	
	Accuracy	F1 (macro)	Accuracy	F1 (macro)
Average	*n.a.*		0.9275	0.9272
Multiply	*n.a.*		0.9332	0.9331
Logistic Regression (C=1)	0.9347	0.9348	0.9338	0.9338
Linear SVM (C=10)	0.9329	0.9328	0.9335	0.9333
Rbf SVM (C=100)	0.9345	0.9346	0.9351	0.9350

[1] Performance CV denotes the average classification performance obtained in the 5-fold cross validation. As the Average and Multiply method do not have hyper-parameters, no cross-validated performance measures are given for these meta-classifiers.

[2] Performance OOS denotes the classification performance when scored on the quasi out-of-sample test set.

6 Conclusion and discussion

The goal of this thesis was to classify financial documents using both textual and visual features, studying what is the best way to use both types of features. In order to do so, documents were first classified using one of the two features types only. For text, this was done using tf-idf as feature extraction method, combined with a linear classifier. For visual input, features were extracted using a pretrained (deep) CNN and classified using a (shallow) fully connected neural network. Moreover, this thesis proposes to combine textual and visual features by means of decision-level fusion. This implies stacking models, using the predicted class probabilities of the textual and visual classification models as input for a meta-classifier.

The main findings of this thesis are the following. First of all, it was found that classification using both textual and visual document information is often superior than using only the one or the other. This was especially seen for the RVL-CDIP data set, where the combined approach yielded the best classification performance to date, outperforming advanced classification methods that used only visual features. Besides, it is seen that good classification performance already can be obtained using only textual features, applying the simple tf-idf feature extraction method and a linear classifier. Although tf-idf brings a relatively large number of hyperparameters that have to be set, their influence is found to be modest. Last of all, when using only visual features for document classification, is it seen that CNNs in combination with transfer learning yield good classification performance. By using transfer learning one can leverage features learned from (for example) the ImageNet data set, which also proves to be beneficial in the context of document (image) classification. However, visual classification techniques are computationally more expensive than the textual methods and require a considerable amount of training data to be available.

In this thesis, a data set was used that was made available by the Rabobank, containing a little over 11,000 labeled observations. However, this data set proved to be both relatively easy to classify and can be considered only of moderate size, which is a limitation for this research. In order to overcome this limitation, the proposed methods were also applied to the RVL-CDIP data set. Although we consider this data set to be comparable, the different data sources have to be kept in mind when generalizing (preliminary) results. Ideally, it would have been preferable to replicate our complete methodology for both data sets, but this was computationally not feasible. Not having more available computational resources is a second limitation of this research. Therefore, we could not (fully) replicate more advanced visual models, that could have potentially increased the visual and overall classification performance even further.

Finally, this thesis can form the basis for further research into the combination of different types of (document) features. While this thesis applied decision-level fusion, further research could focus on feature-level fusion or perhaps on novel combination techniques. Moreover, this work used only one feature extraction method per type of feature. Future research could extend this, focusing on whether different feature extraction techniques can extract complementary information from the same type of features.

References

Afzal, M. Z., Kölsch, A., Ahmed, S., & Liwicki, M. (2017). Cutting the error by half: Investigation of very deep cnn and advanced training strategies for document image classification. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, (pp. 883–888). IEEE.

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, *5*, 135–146.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010* (pp. 177–186). Springer.

Bratko, A. & Filipič, B. (2006). Exploiting structural information for semi-structured document categorization. *Information Processing & Management*, *42*(3), 679–694.

Chen, N. & Blostein, D. (2007). A survey of document image classification: problem statement, classifier architecture and performance evaluation. *International Journal of Document Analysis and Recognition (IJDAR)*, *10*(1), 1–16.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, *12*(Aug), 2493–2537.

Cortes, C. & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297.

Coussement, K. & Van den Poel, D. (2008). Churn prediction in subscription services: An application of support vector machines while comparing two parameter-selection techniques. *Expert Systems with Applications*, *34*(1), 313–327.

Das, A., Roy, S., Bhattacharya, U., & Parui, S. K. (2018). Document image classification with intra-domain transfer learning and stacked generalization of deep convolutional neural networks. In *2018 24th International Conference on Pattern Recognition (ICPR)*, (pp. 3180–3185). IEEE.

Defazio, A., Bach, F., & Lacoste-Julien, S. (2014). Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems*, (pp. 1646–1654).

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (pp. 248–255). IEEE.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, *12*(Jul), 2121–2159.

Dumais, S., Platt, J., Heckerman, D., & Sahami, M. (1998). Inductive learning algorithms and representations for text categorization. In *7th International Conference on Information and Knowledge Management* (7th International Conference on Information and Knowledge Management ed.)., (pp. 148–152).

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. Cambridge, MA: MIT Press.

Govindan, V. & Shivaprasad, A. (1990). Character recognition-a review. *Pattern Recognition*, *23*(7), 671–683.

Harley, A. W., Ufkes, A., & Derpanis, K. G. (2015). Evaluation of deep convolutional nets for document image classification and retrieval. In *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*, (pp. 991–995). IEEE.

Hinton, G. E., McClelland, J. L., Rumelhart, D. E., et al. (1984). *Distributed representations*. Pittsburgh, PA: Carnegie-Mellon University.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. In *European Conference on Machine Learning*, (pp. 137–142). Springer.

Joulin, A., Grave, E., Bojanowski, P., & Mikolov, T. (2017). Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, (pp. 427–431)., Valencia, Spain. Association for Computational Linguistics.

Kingma, D. P. & Ba, J. (2014). Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*.

Lancaster, F. W. (1978). *Towards paperless information systems*. Cambridge, MA: Academic Press.

Lang, K. (1995). Newsweeder: learning to filter netnews. In *Proceedings of the Twelfth International Conference on International Conference on Machine Learning*, (pp. 331–339). Morgan Kaufmann Publishers Inc.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, *1*(4), 541–551.

Lewis, D., Agam, G., Argamon, S., Frieder, O., Grossman, D., & Heard, J. (2006). Building a test collection for complex document information processing. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, (pp. 665–666). ACM.

Li, M., Zhang, T., Chen, Y., & Smola, A. J. (2014). Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, (pp. 661–670). ACM.

Loper, E. & Bird, S. (2002). Nltk: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, (pp. 63–70). Association for Computational Linguistics.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (2011). Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-volume 1*, (pp. 142–150). Association for Computational Linguistics.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. In *ICLR: Proceeding of the International Conference on Learning Representations Workshop Track*, (pp. 1301–3781).

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, (pp. 3111–3119).

Nair, V. & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, (pp. 807–814).

Nielsen, M. A. (2015). *Neural Networks and Deep Learning*, volume 25. San Francisco, CA: Determination Press.

Pelka, O. & Friedrich, C. M. (2015). Fhdo biomedical computer science group at medical classification task of imageclef 2015. In *CLEF (Working Notes)*, volume 1391. CEUR Workshop Proceedings.

Platt, J. et al. (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in Large Margin Classifiers*, *10*(3), 61–74.

Poria, S., Cambria, E., Howard, N., Huang, G.-B., & Hussain, A. (2016). Fusing audio, visual and textual clues for sentiment analysis from multimodal content. *Neurocomputing*, *174*, 50–59.

Raschka, S. (2018). Model evaluation, model selection, and algorithm selection in machine learning. *arXiv preprint arXiv:1811.12808*. Retrieved August 16, 2019, from `https://arxiv.org/abs/1811.12808`.

Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. (1988). Learning representations by back-propagating errors. *Cognitive Modeling*, *5*(3), 1.

Salton, G. & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information Processing & Management*, *24*(5), 513–523.

Salton, G., Wong, A., & Yang, C.-S. (1975). A vector space model for automatic indexing. *Communications of the ACM*, *18*(11), 613–620.

Sebastiani, F. (2002). Machine learning in automated text categorization. *ACM Computing Surveys (CSUR)*, *34*(1), 1–47.

Shan, C., Gong, S., & McOwan, P. W. (2007). Beyond facial expressions: Learning human emotion from body gestures. In *BMVC: Proceedings of the British Machine Vision Conference 2007*, (pp. 1–10).

Simonyan, K. & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*.

Smith, R. (2007). An overview of the tesseract ocr engine. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 2, (pp. 629–633). IEEE.

Sumathy, K. & Chidambaram, M. (2013). Text mining: concepts, applications, tools and issues-an overview. *International Journal of Computer Applications*, *80*(4).

Taghva, K., Borsack, J., & Condit, A. (1996). Effects of ocr errors on ranking and feedback using the vector space model. *Information Processing & Management*, *32*(3), 317–327.

Tensmeyer, C. & Martinez, T. (2017). Analysis of convolutional neural networks for document image classification. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, (pp. 388–393). IEEE.

Tieleman, T. & Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, *4*(2), 26–31.

Uysal, A. K. & Gunal, S. (2014). The impact of preprocessing on text classification. *Information Processing & Management*, *50*(1), 104–112.

Van Rijsbergen, C. (1986). A new theoretical framework for information retrieval. In *Proceedings of the 9th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, (pp. 194–200). ACM.

Villegas, M., Müller, H., Gilbert, A., Piras, L., Wang, J., Mikolajczyk, K., de Herrera, A. G. S., Bromuri, S., Amin, M. A., Mohammed, M. K., et al. (2015). General overview of imageclef at the clef 2015 labs. In *International Conference of The Cross-language Evaluation forum for European languages*, (pp. 444–461). Springer.

Yang, Y. & Liu, X. (1999). A re-examination of text categorization methods. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, (pp. 42–49). ACM.

Zhang, X., Zhao, J., & LeCun, Y. (2015). Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*, (pp. 649–657).

Appendix A - Rabobank

Table 10: Classification report optimal linear SVM model (text)

	Class name	Precision	Recall	F1-score	Support
2	Zekerhedenregeling	1.0000	0.9286	0.9630	14
4	Borgtocht	0.9412	1.0000	0.9697	16
6	Lease-overeenkomst	0.9870	0.9769	0.9819	389
8	Kredietgoedkeuring	0.9458	0.9522	0.9490	293
15	Overige documenten Lease	0.9811	0.9630	0.9720	54
16	Hoofdelijke medeschuldverbintenis	1.0000	0.5000	0.6667	6
17	Aanhangsel lease	1.0000	1.0000	1.0000	3
18	Leasevoorwaarden	0.8605	0.9487	0.9024	39
19	Lease-offerte	0.7955	0.8537	0.8235	41
20	SEPA-machtigingsformulier	1.0000	1.0000	1.0000	5
21	Overige correspondentie klant	0.9854	0.9543	0.9696	635
23	Jaarcijfers	0.9633	0.9459	0.9545	111
24	Checklist	0.9956	0.9784	0.9869	231
25	Uitboeking	0.9643	0.9474	0.9558	114
26	KVK extract	0.8769	0.9048	0.8906	63
28	Factuur	0.9490	0.9907	0.9694	432
29	Betaalbewijs lease-object	0.8542	0.9762	0.9111	42
	Accuracy			0.9622	2488
	Macro Avg.	0.9470	0.9306	0.9333	2488
	Weighted Avg.	0.9634	0.9622	0.9623	2488

(Optimal) hyperparameters used: unigrams and bigrams included, min. term document frequency (df) = 2, max. term df = *None*, max. features number = *None*, normalization used: l_2, term frequency type: logarithmically scaled, SVM regularization constant $C = 1$.

Table 11: Classification report optimal logistic regression model (text)

	Class name	Precision	Recall	F1-score	Support
2	Zekerhedenregeling	1.0000	0.9286	0.9630	14
4	Borgtocht	0.9375	0.9375	0.9375	16
6	Lease-overeenkomst	0.9794	0.9769	0.9781	389
8	Kredietgoedkeuring	0.9360	0.9488	0.9424	293
15	Overige documenten Lease	0.9804	0.9259	0.9524	54
16	Hoofdelijke medeschuldverbintenis	1.0000	0.5000	0.6667	6
17	Aanhangsel lease	1.0000	0.6667	0.8000	3
18	Leasevoorwaarden	0.8605	0.9487	0.9024	39
19	Lease-offerte	0.7907	0.8293	0.8095	41
20	SEPA-machtigingsformulier	1.0000	1.0000	1.0000	5
21	Overige correspondentie klant	0.9791	0.9591	0.9690	635
23	Jaarcijfers	0.9633	0.9459	0.9545	111
24	Checklist	1.0000	0.9784	0.9891	231
25	Uitboeking	0.9818	0.9474	0.9643	114
26	KVK extract	0.8769	0.9048	0.8906	63
28	Factuur	0.9532	0.9907	0.9716	432
29	Betaalbewijs lease-object	0.8696	0.9524	0.9091	42
	Accuracy			0.9606	2488
	Macro Avg.	0.9476	0.9024	0.9177	2488
	Weighted Avg.	0.9615	0.9606	0.9606	2488

(Optimal) hyperparameters used: unigrams and bigrams included, min. term document frequency (df) = 5, max. term df = 0.5, max. features number = 20,000, normalization used: l_2, term frequency type: logarithmically scaled, logistic regression constant $C = 10$.

Table 12: Classification report optimal segmentation model (text)

	Class name	Precision	Recall	F1-score	Support
2	Zekerhedenregeling	0.9333	1.0000	0.9655	14
4	Borgtocht	1.0000	0.9375	0.9677	16
6	Lease-overeenkomst	0.9845	0.9769	0.9806	389
8	Kredietgoedkeuring	0.9426	0.9522	0.9474	293
15	Overige documenten Lease	0.9811	0.9630	0.9720	54
16	Hoofdelijke medeschuldverbintenis	1.0000	0.6667	0.8000	6
17	Aanhangsel lease	1.0000	1.0000	1.0000	3
18	Leasevoorwaarden	0.8837	0.9744	0.9268	39
19	Lease-offerte	0.7955	0.8537	0.8235	41
20	SEPA-machtigingsformulier	1.0000	1.0000	1.0000	5
21	Overige correspondentie klant	0.9762	0.9669	0.9715	635
23	Jaarcijfers	0.9902	0.9099	0.9484	111
24	Checklist	0.9956	0.9784	0.9869	231
25	Uitboeking	0.9732	0.9561	0.9646	114
26	KVK extract	0.8750	0.8889	0.8819	63
28	Factuur	0.9572	0.9838	0.9703	432
29	Betaalbewijs lease-object	0.8913	0.9762	0.9318	42
	Accuracy			0.9634	2488
	Macro Avg.	0.9517	0.9403	0.9435	2488
	Weighted Avg.	0.9642	0.9634	0.9635	2488

(Optimal) hyperparameters used: unigrams and bigrams included, min. term document frequency (df) = 2, max. term df = *None*, max. features number = *None*, normalization used: l_2, term frequency type: logarithmically scaled, SVM regularization constant C=1. Segmentation used: $(X, Y) = (4, 1)$.

Table 13: Classification report optimal CNN model (visual)

	Class name	Precision	Recall	F1-score	Support
2	Zekerhedenregeling	0.7500	0.6429	*0.6923*	14
4	Borgtocht	1.0000	0.6250	*0.7692*	16
6	Lease-overeenkomst	0.9223	0.9769	0.9488	389
8	Kredietgoedkeuring	0.8736	0.8020	0.8363	293
15	Overige documenten Lease	0.9615	0.9259	0.9434	54
16	Hoofdelijke medeschuldverbintenis	1.0000	0.1667	*0.2857*	6
17	Aanhangsel lease	1.0000	0.6667	*0.8000*	3
18	Leasevoorwaarden	0.8333	0.8974	0.8642	39
19	Lease-offerte	0.6400	0.7805	0.7033	41
20	SEPA-machtigingsformulier	1.0000	1.0000	1.0000	5
21	Overige correspondentie klant	0.9200	0.9417	0.9307	635
23	Jaarcijfers	0.9512	0.7027	0.8083	111
24	Checklist	0.9824	0.9654	0.9738	231
25	Uitboeking	0.9722	0.9211	0.9459	114
26	KVK extract	0.7467	0.8889	*0.8116*	63
28	Factuur	0.9124	0.9398	0.9259	432
29	Betaalbewijs lease-object	0.8696	0.9524	0.9091	42
	Accuracy			0.9104	2488
	Macro Avg.	0.9021	0.8115	0.8323	2488
	Weighted Avg.	0.9128	0.9104	0.9090	2488

Model architecture used: # of layers = 1, dimensionality size = 1024.

Table 14: Classification report linear SVM combined features model

	Class name	Precision	Recall	F1-score	Support
2	Zekerhedenregeling	0.8667	0.9286	0.8966	14
4	Borgtocht	0.9333	0.8750	0.9032	16
6	Lease-overeenkomst	0.9921	0.9692	0.9805	389
8	Kredietgoedkeuring	0.9488	0.9488	0.9488	293
15	Overige documenten Lease	0.9808	0.9444	0.9623	54
16	Hoofdelijke medeschuldverbintenis	1.0000	0.5000	0.6667	6
17	Aanhangsel lease	1.0000	1.0000	1.0000	3
18	Leasevoorwaarden	0.8605	0.9487	0.9024	39
19	Lease-offerte	0.8085	0.9268	0.8636	41
20	SEPA-machtigingsformulier	1.0000	1.0000	1.0000	5
21	Overige correspondentie klant	0.9747	0.9701	0.9724	635
23	Jaarcijfers	0.9722	0.9459	0.9589	111
24	Checklist	1.0000	0.9784	0.9891	231
25	Uitboeking	0.9818	0.9474	0.9643	114
26	KVK extract	0.8769	0.9048	0.8906	63
28	Factuur	0.9640	0.9907	0.9772	432
29	Betaalbewijs lease-object	0.8511	0.9524	0.8989	42
	Accuracy			0.9642	2488
	Macro Avg.	0.9418	0.9254	0.9280	2488
	Weighted Avg.	0.9654	0.9642	0.9643	2488

(Optimal) hyperparameters used: Model based on optimal hyperparameters found in textual and visual models. Regularization constant of linear SVM $C = 1$.

Appendix B - RVL-CDIP

Table 15: Classification report optimal textual linear SVM model ($C = 1$)

	Class name	Precision	Recall	F1-score	Support
0	Letter	0.8899	0.8231	0.8552	2464
1	Form	0.8789	0.7997	0.8374	2506
2	Email	0.9721	0.9424	0.9570	2516
3	Handwritten	0.7813	0.7800	0.7806	2532
4	Advertisement	0.8447	0.7332	0.7850	2515
5	Scientific report	0.8355	0.7950	0.8148	2498
6	Scientific publication	0.8901	0.9257	0.9075	2571
7	Specification	0.9463	0.9405	0.9434	2472
8	File folder	0.6102	0.8896	0.7239	2527
9	News article	0.8686	0.8372	0.8526	2463
10	Budget	0.9071	0.8850	0.8959	2505
11	Invoice	0.9138	0.8946	0.9041	2477
12	Presentation	0.8450	0.8433	0.8442	2489
13	Questionnaire	0.9124	0.9281	0.9202	2435
14	Resume	0.9817	0.9724	0.9770	2537
15	Memo	0.8828	0.8435	0.8627	2492
	Accuracy			0.8646	39999
	Macro Avg.	0.8725	0.8646	0.8663	39999
	Weighted Avg.	0.8723	0.8646	0.8662	39999

Hyperparameters used: unigrams and bigrams included, min. term document frequency (df) = 5, max. term df = 0.5, max. features number = $2,000,000$, normalization used: l_2, term frequency type: logarithmically scaled, linear SVM regularization constant $C = 1$.

Table 16: Classification report optimal textual logistic regression model ($C = 10$)

	Class name	Precision	Recall	F1-score	Support
0	Letter	0.8878	0.8190	0.8520	2464
1	Form	0.8752	0.8005	0.8362	2506
2	Email	0.9720	0.9380	0.9547	2516
3	Handwritten	0.7749	0.7994	0.7869	2532
4	Advertisement	0.8169	0.7396	0.7763	2515
5	Scientific report	0.8298	0.7942	0.8116	2498
6	Scientific publication	0.8923	0.9218	0.9068	2571
7	Specification	0.9499	0.9349	0.9423	2472
8	File folder	0.6232	0.8908	0.7333	2527
9	News article	0.8622	0.8356	0.8487	2463
10	Budget	0.9020	0.8822	0.8920	2505
11	Invoice	0.9105	0.8870	0.8986	2477
12	Presentation	0.8359	0.8309	0.8334	2489
13	Questionnaire	0.9135	0.9199	0.9167	2435
14	Resume	0.9816	0.9669	0.9742	2537
15	Memo	0.8834	0.8331	0.8575	2492
	Accuracy			0.8622	39999
	Macro Avg.	0.8694	0.8621	0.8638	39999
	Weighted Avg.	0.8692	0.8622	0.8637	39999

Hyperparameters used: unigrams and bigrams included, min. term document frequency (df) = 5, max. term df = 0.5, max. features number = $2,000,000$, normalization used: l_2, term frequency type: logarithmically scaled, logistic regression regularization constant $C = 10$.

Table 17: Classification report best visual model

	Class name	Precision	Recall	F1-score	Support
0	Letter	0.8367	0.8819	0.8587	2464
1	Form	0.7566	0.7889	0.7724	2506
2	Email	0.9589	0.9817	0.9701	2516
3	Handwritten	0.9270	0.9431	0.9350	2532
4	Advertisement	0.9141	0.8974	0.9057	2515
5	Scientific report	0.7546	0.7314	0.7428	2498
6	Scientific publication	0.9384	0.9004	0.9190	2571
7	Specification	0.9202	0.9001	0.9100	2472
8	File folder	0.8975	0.9632	0.9292	2527
9	News article	0.8893	0.9001	0.8947	2463
10	Budget	0.8481	0.8467	0.8474	2505
11	Invoice	0.8963	0.8724	0.8842	2477
12	Presentation	0.7863	0.8027	0.7944	2489
13	Questionnaire	0.8338	0.8012	0.8172	2435
14	Resume	0.9542	0.9279	0.9408	2537
15	Memo	0.9128	0.8780	0.8951	2492
	Accuracy			0.8764	39999
	Macro Avg.	0.8765	0.8761	0.8761	39999
	Weighted Avg.	0.8769	0.8764	0.8764	39999

Model architecture used: Feature extraction using first four convolutional blocks of VGG16, followed by training 3 Convolutional layers $3 \times 3 \times 512$, a max pooling layer 2×2, 2 fully connected layers of size 4096 + softmax 16 classes.

Table 18: Classification report rbf kernel SVM combined features model

	Class name	Precision	Recall	F1-score	Support
0	Letter	0.9107	0.9188	0.9147	2464
1	Form	0.8652	0.8603	0.8627	2506
2	Email	0.9791	0.9869	0.9830	2516
3	Handwritten	0.9560	0.9526	0.9543	2532
4	Advertisement	0.9490	0.9392	0.9440	2515
5	Scientific report	0.8783	0.8695	0.8739	2498
6	Scientific publication	0.9539	0.9502	0.9521	2571
7	Specification	0.9574	0.9555	0.9565	2472
8	File folder	0.9658	0.9731	0.9694	2527
9	News article	0.9269	0.9269	0.9269	2463
10	Budget	0.9331	0.9250	0.9290	2505
11	Invoice	0.9393	0.9370	0.9382	2477
12	Presentation	0.8810	0.9044	0.8925	2489
13	Questionnaire	0.9366	0.9466	0.9416	2435
14	Resume	0.9905	0.9823	0.9863	2537
15	Memo	0.9374	0.9318	0.9346	2492
	Accuracy			0.9351	39999
	Macro Avg.	0.9350	0.9350	0.9350	39999
	Weighted Avg.	0.9352	0.9351	0.9351	39999

Optimal) hyperparameters used: Model based on (optimal) hyperparameters found in textual and visual models. Regularization constant of RBF kernel SVM $C = 100$.