Vrije Universiteit Amsterdam

Stater N.V.

Master Thesis

Splitting multi-document files into logical sub-documents using multimodal deep learning models

Author: Jens van Holland (2651896)

1st supervisor: Dr. Jakub Tomczak
daily supervisor: Dr. Guozhong An (Stater N.V)
2nd reader: Dr. Florian Kunneman

A thesis submitted in fulfilment of the requirements for the Vrije Universiteit Amsterdam Master of Science degree in Business Analytics.

September 6, 2022

"Machines take me by surprise with great frequency"
from Computing Machinery and Intelligence (1950), by Alan Turing

Preface

This thesis is written for the Master of Science degree in Business Analytics at the Vrije Universiteit Amsterdam. The main focus of this thesis is to research on how to split a file into logical sub-documents using automatically extracted features from textual and graphical page representations. Stater N.V. provided the data and resources to research this problem. Stater N.V. is a Dutch mortgage administration company, it manages the mortgage administration for several moneylenders (e.g. banks). The main motivation for this research is my personal interest in deep learning architectures and their applications in Natural Language Processing and Computer Vision type of use-cases. I would like to thank the Datalab team at Stater N.V. for their help and insights, especially my daily supervisor Gouzong An. Also, I would like to thank Jakub Tomczak for the monthly meetings about my research progress and the feedback on my thesis. At last, I appreciate Florian Kunneman for reading and grading my thesis as my second reader.

Abstract

Context. The Dutch mortgage market generates a lot of unstructured data. Stater N.V. is a large administration company which manages the mortgage administration of several moneylenders. A lot of unstructured data is stored in digital archives, such as documents, photos and e-mails. Stater N.V., and its clients, aim to bring more structure to the data in these digital archives. The aim of this thesis is to research methods for splitting and classifying Dutch multi-document mortgage files. This will contribute to structuring the archive. The data consists of PDF files. Since not every file can be read out by a PDF reader, e.g. because they consist of scanned pages, an Optical Character Recognition (OCR) engine is used. To select a fitting OCR model for the provided data, several off-the-shelf OCR models are tested on benchmark document data sets.

Goal. The intermediate goal is to select a fitting OCR model which works well on (scanned) documents to extract the textual data. The main goal is to develop a model which can split a file, of any given length, into multiple documents. Each extracted sub-document is also classified with a document type.

Methods. Two types of models are used. Firstly, a Support Vector Machine (SVM), which makes page-wise classifications using solely textual data. These classifications are then used for determining the splits in a file. The second type of approach is to use neural network architectures (Time Distributed Convolutional and Recurrent Neural Networks) which directly optimise where to split from textual and graphical context information, and classify each page with a page type. The difference in approach is that the SVM only uses the textual data, while the neural networks use textual and graphical data, i.e. multimodality. Also, the split is indirectly optimised by the SVM, in contrast to the networks, which optimise it directly. The files are split into three sets for training, validation and testing purposes. For each set, documents are (pseudo)randomly merged together and used to train and validate the models. Since the primary interest is to see what the performance of a model is on a file, three metrics are used on the evaluation sets: (1) the mean percentage of correctly made splits/no splits per file, (2) the mean percentage of correctly classified pages per file and (3) the mean percentage of fully correctly extracted sub-documents per file.

Results. The results show that the SVM model works best on the classification (2) and sub-document extraction (3), with scores of 92.14% and 85.09% on the test evaluation sets, respectively. The neural network models only perform slightly worse on these metrics: the TDCNN-RNN$^{(2:2)}$ got 91.68% on the classification task and a the RNN$^{(1:1)}$ got 84.33% on the sub-document extraction. However, the TDCNN-RNN$^{(3:3)}$ does perform best on the splitting (1) with a 96.87%. The SVM splits 93.06% correctly. In general, the networks which only have access to the graphical features have the lowest performance. Also, adding more context, in terms of graphical and textual page information, does not seem to have a positive effect on the performance of the models. The error analysis shows that the SVM and TDCNN-RNN$^{(3:3)}$ model seem to have the most confusion between the gold label *No split* and predicted label *Split*. This is surprising, since most multi-document files have more pages than sub-documents, thus require more *No split* than *Split*. So while the models are trained, and evaluated, on data sets with a majority class of *No split*, they do tend to split files more often than necessary.

Conclusions. Based on the results and error analysis, it seems that using textual features with a SVM does work best. Although the neural network models do not outperform the SVM on all metrics, there are indicators that directly optimising the split classification works better than indirectly optimising. The neural networks, with a vision part, do seem to look at visually rich areas (in the first convolutional layer) such as company logos and emblems. In future work, the scalability of a solution should be investigated, since the archive contains billions of pages which all need to be processed.

Contents

List of Figures

LIST OF FIGURES

List of Tables

1

Introduction

1.1 Research questions

The main research question is as follows,

To what extent can Convolutional and Recurrent Neural Networks be used to partition Dutch mortgage files based on their graphical and textual content into separate logical mortgage documents?

To answer this research question, five sub-questions are formulated, namely,

- *How can this problem be formulated as a general solvable problem?*
  The problem and data as provided are not that common in similar research, so one needs to formulate it as a more general and traditional solvable problem, i.e. categorical classification problems. The problem is mathematically formulated in Chapter 3.

- *What Optical Character Recognition methods can be used to gather text from non-readable files?*
  The original data are stored as PDFs, for a large portion of these files one can simply use a PDF text reader to extract the textual data. For a smaller portion of the files a reader is not sufficient to extract the textual data (i.e. a document is scanned or consists of images). One type of method to extract these textual data is called Optical Character Recognition (OCR). To research which OCR method performs best (from a selection of chosen models), a benchmark is done with benchmark data sets in Chapter 4.

- *What effect does the synthetic data have on the model performance?*
  As explained in Chapter 5, documents are (conditionally) randomly merged together to create synthetic evaluation sets. To see what effect this random merging has on the performance of the model, the results (Chapter 6) are averaged over the evaluation sets. The standard deviations of these results show if the model is robust against randomly merged sets.

- *What type of Deep Learning architectures can be used for this problem?*
  There are several types of Deep Learning layers and units (i.e. dense, convolutional and recurrent) which can be used to process data. To see what type and variants of these layers are effective for this particular problem, several architectures are tried. The chosen architectures are discussed and experimented with in Chapter 3 and 6.

- *Which parts of the input data does the model base its decisions on?*
  Deep Learning models are so-called black box models, meaning that it is not (precisely) known how a model inferred a particular outcome from the input. To make a model less black box, a qualitative analysis is done on the model output to see if logical parts of the input are used to make decisions. This is done with Grad-CAM [52] in Chapter 3.

The answers to the main research question and sub-questions are discussed in Chapter 7.

2

Related work

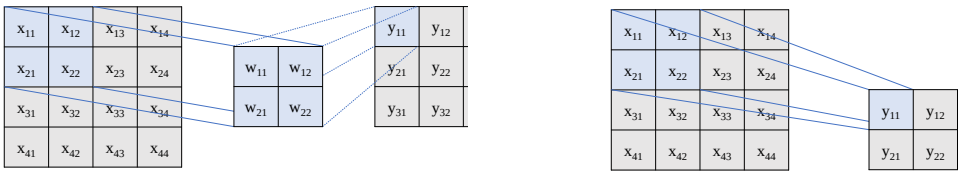
In this chapter, the relevant literature is discussed. The relevant fields for this research report are neural networks, computer vision and document understanding.

2.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) can be described as a combination of simple connected processing units (neurons) which communicate by sending signals to each other and which can collectively learn from an environment (e.g. a data set) using some learning process [26, 57]. The ANN tries to minimise an error function (e.g. crossentropy, mean squared error) by adjusting the weights inside the network, which is usually done incrementally. This type of learning is done with backpropagation, i.e. the error made by the ANN goes back through the network to let the model learn from its mistakes. Neural networks are mainly used for classification, regression and clustering. There are several types of ANN layers, such as dense / fully connected, transformer, convolutional and recurrent layers. The last two mentioned are explained in more detail in the next paragraphs.

2.1.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are neural networks which are specialised in processing grid-like data, such as time-series and images [23]. A CNN consists of convolutions, also called filters, which usually map the input (e.g. a 2D image) into a smaller sized output, as shown in Figure 2.1 (a). The output of each filter applied to the input is also called a feature map. These filters contain trainable parameters, denoted as w in Figure 2.1 (a). In practice, one uses multiple filters simultaneously which is referred to as a convolutional layer. A convolutional layer (or group of convolutional layers) is usually followed by a max,



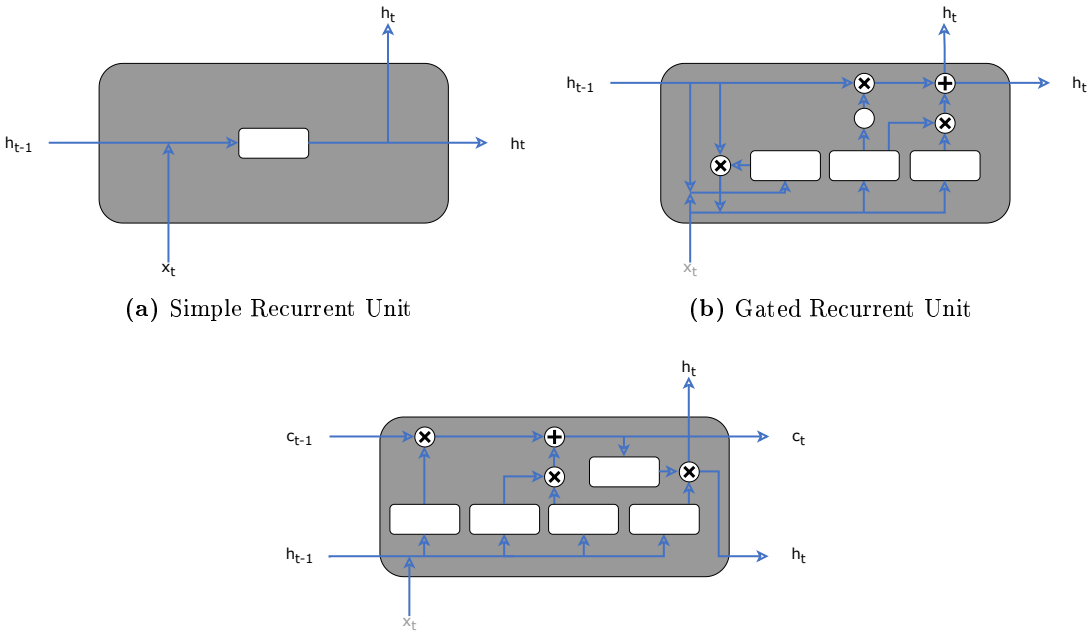
(a) Convolutional filter **(b)** Pooling filter

Figure 2.1: An abstract example of a convolutional filter (kernel size: 2 x 2, stride: 1) and a pooling filter (kernel size: 2 x 2, stride: 2)

min or average pooling layer. This pooling layer selects or computes a value from a grid of candidate values (i.e. a pool) which make up a new, and usually smaller, grid. As one can see in Figure 2.1 (b), a pooling with a 2 by 2 grid and a stride of 2 is applied on a 4 by 4 input. This results in a 2 by 2 output. After a convolutional or pooling layer, a non-linear activation function, such as the ReLU or tanh, is commonly used. These non-linearities allow the network to make a non-linear decision space. These convolutional layer - pooling layer - activation layer blocks are usually stacked to allow the model to learn more abstract relationships and dependencies. Examples of well known network architectures are ResNet [29], VGG-16 [56], MobileNet (v3) [32] and Inception (v2) [35].

2.1.2 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are models which are able to selectively pass information over (variable) sequence steps [41]. An RNN takes in a sequence (e.g. a timeseries) and produces a sequence. A sequence contains one or more values, for both the input and produced output of an RNN. Thus, an RNN can be seen as a loop where each input (and/or produced output) is inserted into the RNN. This means that an RNN shares its parameters with each step in the sequence, instead of assigning separate parameters for each step in the sequence. This means that the model has relatively fewer parameters to learn. In contrast to a CNN, which can backpropagate the errors in parallel, an RNN needs to backpropagate the error using the same loop as the forward propagation, resulting in a relatively longer learning time. One of the first RNN units was the Simple RNN or Elman RNN [20], this type of network takes the output of the previous step and the input of the current step. Then an activation function, such as the ReLU or tanh, is used to produce



(a) Simple Recurrent Unit

(b) Gated Recurrent Unit

(c) Long Short-Term Memory Unit

Figure 2.2: Examples of three Recurrent Neural Network Units: (a) Simple Recurrent Unit / Elman Recurrent Unit, (b) Gated Recurrent Unit and (c) Long Short-Term Memory Unit.

an output. This output is used as input for the next layer or next time step. An example of the Elman RNN unit is shown in Figure 2.2 (a). The main problems with this type of RNN are the vanishing and exploding gradients when learning long term dependencies in a sequence [13]. To counter this problem, a new family of RNN units was developed, namely, the Gated Recurrent Neural Networks. One of these units is the Long Short-Term Memory (LSTM) [25, 31], the internal structure of an LSTM unit is shown in Figure 2.2 (c). An LSTM is able to filter or select information using its internal gates, thus is able to detect and keep information of long range dependencies [13]. Another type of gated unit is the Gated Recurrent Unit (GRU) [12, 13]. This unit also contains several gates to select and filter information as can be seen in Figure 2.2 (b).

2.2 Computer Vision

Computer Vision is the computer science field of automatically processing, seeing and understanding the visual world by machines [21]. The tasks in this field include, image classification, object recognition, object detection and image retrieval. More recently, deep learning has a large presence on the development of computer vision algorithms [59]. This

can also be seen in the current state-of-the-art computer vision models, such as [4, 42, 66], on benchmark data sets such as MNIST[16], COCO[40] and SIDD[1]. A special case of computer vision is Optical Character Recognition, this case of computer vision is a science which researches how to translate text on an image (i.e. pixel-ed symbols) to real text (i.e. typed symbols).

2.2.1 Optical Character Recognition

Optical Character Recognition (OCR) is the field where symbols (i.e. text) is extracted from an image. These images can be natural images, also referred to as images in the wild/camera-based, or scanned document pages (e.g. PDF). There are some differences between these two types of images, discussed by [11, 51], such as text density, complexity of the background and noise level, size, colour and rotation of the characters. An end-to-end OCR application has usually two main components, namely, text detection and text recognition [18, 33, 38, 43]. Text detection is the study about detecting text (i.e. characters and symbols) in an image. This field is part of the larger object detection research landscape, but has several different characteristics and therefore needing its own custom solutions and methods [43]. Text recognition is the field of extracting text from an image. Thus, the text detection algorithm passes the isolated potential text regions to the text recognition algorithm, which converts the images to plain text.

2.3 Document Understanding

Document Understanding, or Document Intelligence, is a relatively new field of research which aims to analyse, recognise, extract, classify and structuralise information from digital unstructured documents [2, 46]. Optical Character Recognition methods can be seen as part of the document understanding field. Documents which contain a noticeable layout are typically mentioned in literature as Visual Rich Documents (VRDs). Document understanding also leverages the document layout instead of using only the (text) content. Thus based on the graphical and textual features, an algorithm tries to understand a document [5, 63]. Next, pre-trained transformer models and transfer learning are widely used in NLP. Most of these pre-trained models, such as BERT [17] or BERT like models, are purely text based and discard or ignore layout and style features, while these can be very useful to represent text semantics [64]. The LayoutLM [64], LayoutLMv2 [63], LayoutLMv3 [34] and LayoutXLM [65] models use the text but also the text positions (i.e.

layout) and visual information from VRDs. These models contain some form of transformers (e.g. BERT backbone) and are trained on millions of documents. Next, the models are finetuned on a downstream document understanding task such as Form Understanding, Document Classification and Receipt Understanding with benchmark data sets. The results from [63, 64, 65] show that textual and layout based models perform better than purely textual based models on several tasks.

2.3.1 Page Stream Segmentation

A task in document understanding is Page Stream Segmentation (PSS). PSS aims to segmentate, i.e. split, a multipage document into logical sub-documents. In literature [14, 24], page text and layout features are used to effectively segmentate documents. Examples of these features are the page number of a page, words, line spacing/indentation and header/footer information. The authors from [14] use a bottom-up clustering approach; each page is in its own cluster and clusters are merged together based on a distance metric. Using these clusters, one could segmentate the original document in sub-documents. Another method is presented by [15], the authors used regular expressions to gather specific textual information of administrative documents pages. These regular expressions extracted features such as the date, hour, telephone number, page number and postcode. Using these extracted features, they compared two successive pages and made a new type of feature vector based on the similarity of previously extracted features. This way they engineered features representing relationships between two successive pages. Next, the authors of [3] engineered features based on (binary) page images, such as the pixel column/row standard deviations and pixel transition intensities. Also, they used a Bag of Visual Words (BoVW) model. A BoVW consists of "visual words", these type of words are usually presented as vectorized patches of an image. A well-known method to vectorize these patches is Scale-Invariant Feature Transform (SIFT) [45], each patch is represented by 128 values. Next, similar vectors are grouped by a clustering method and a representative vector is computed (e.g. cluster mean). So, the vocabulary of a BoVW model consists of cluster representative vectors. As with the NLP Bag of Words model, one counts the number "visual words" in an image and present it as a (sparse) vector. Using these features, the authors of [3] use Support Vector Machines (SVMs), Random Decision Forests (RDFs) and Multilayer Perceptrons (MLPs) to classify if two pages are of the same document or not.

2.3.2 Document classification

Another task of document understanding is Document Classification (DC). This task aims to predict the type of a document with an algorithm using the document pages, represented as images and/or text, as input. Several types of methods are mentioned in literature, such as CNN architectures [27, 37], Decision Trees [6, 54, 60] and K-Nearest Neighbour [7, 30]. From [10], it is clear that there are three main types of features used in this kind of research, namely,

- *Image features*
  These features are using the document represented as an image (i.e. pixels). Examples of these type of features are the density of black pixels in a region and the number of horizontal lines and column/row gaps.

- *Layout features*
  A page is segmented into different regions. These extracted regions can have some structural relationship which can be leveraged by an algorithm.

- *Textual features*
  Textual features are usually extracted using an OCR engine or text reader. After the text extraction, the text of each document or page is vectorized using the Term Frequency - Inverse Document Frequency or other counting methods.

Each of these feature types capture different information from a document page, the importance of a feature type differs per document type. For example, layout features work fairly well for forms or cheques. This is because these type of documents are restricted to certain templates or conventions, thus the physical layout is a good predictor for these types of documents.

3

Methodology

In this chapter the (mathematical) problem statement, objectives, solution methods and evaluation methods are discussed.

3.1 Problem statement

The problem at hand is to split a given file (of any length) into logical sub-documents. A logical sub-document is a document which consists of pages with similar textual and/or graphical information and thus belonging to the same document type. The pages of each sub-document are in a consecutive order in the original file i.e. a sub-document is a subset of sequential pages from the original file. In the remainder of this thesis, a file is defined as a sequence of pages of which the page type can, but does not necessarily have to, differ. A (sub-)document is defined as a sequence of pages of which the page types are all the same. In general, a document page usually refers to the content (i.e. textual representation) and layout (i.e. graphical representation) of that page. The research question states two objectives, namely, partitioning a file (i.e. split) into logical mortgage documents (i.e. classification). There are few approaches on how to split (and classify sub-documents of) a file based on graphical and textual page representations. Two methods are discussed in this section.

3.1.1 Formulation I

Firstly, the variables, (hyper)parameters and data are defined and described using the notation below, to start with the data set.

$\mathcal{D}^* :=$ the original data set

The scalar variables are defined as,

$$
\begin{aligned}
N^* &:= \text{ the total number of documents} \\
n_i^* &:= \text{ the number of pages of document } i \\
H &:= \text{ the number of words in the vocabulary} \\
C &:= \text{ the number of page types} \\
w &:= \text{ the page width (in pixel values) of the graphical representation} \\
h &:= \text{ the page height (in pixel values) of the graphical representation}
\end{aligned}
$$

Parts of the data are defined as,

$$
\begin{aligned}
\mathbf{V} &:= \text{ the vocabulary (unique words) of the data set} \\
\mathbf{X}_i^* &:= \text{ the graphical representations of the pages of document } i \\
\mathbf{Y}_i^* &:= \text{ the textual representations of the pages of document } i \\
\mathbf{T}_i^* &:= \text{ the pages types of document } i \\
x_i^{j*} &:= \text{ the graphical representation of page } j \text{ of document } i \\
y_i^{j*} &:= \text{ the textual representation of page } j \text{ of document } i \\
t_i^{j*} &:= \text{ the class of page j of document } i \\
&\text{with } j \in \{1, ..., n_i^*\}, i \in \{1, 2, 3, ..., N^*\}
\end{aligned}
$$

The dimensions and values of the previously mentioned variables and data are,

$$
\begin{aligned}
N^* &= |\mathcal{D}^*| \\
H &= |\mathbf{V}| \\
x_i^{j*} &\in \mathbb{R}^{w \times h \times 3} \\
y_i^{j*} &\in \mathbb{R}^H \\
\mathcal{D}^* &= \{\mathbf{X}_i^*, \mathbf{Y}_i^*, \mathbf{T}_i^* | \forall i \in \{1, 2, 3, ..., N^*\}\}
\end{aligned}
$$

The problem can be categorised as a *multiclass-multioutput classification* or *multitask classification* [50] problem with *multimodal sequential ordered data*, i.e. page images and text, as input. This multiclass-multioutput classification problem formulation expects all the pages of a file as input, and outputs for every page if a certain document type ends

(i.e. split) or if no split is is needed. Thus, for every page, the model should output $C+1$ values, one for each document type and the *no split* class. In total, $n_i^* \cdot (C+1)$ output values need to be provided by the model. It is important to mention a few aspects of this modelling approach:

- The data consists of N^* files, these files do not necessarily contain one document type but can contain several document types.
- The document types can be wildly different. For example, an identification card usually consists of one page while a notary document can contain up to 40 pages. Another example is that some (parts of the) files can be digital and others are scanned in.
- Based on the two points above one can conclude that the files in the original data, $\mathcal{D}^*$, are also of very different lengths, i.e. n_i^* can be wildly different.

So, with this modelling approach, one would take $n_{max} = \max_i(n_i^*)$ as an upper bound for the number of pages a document can have, and train a model which outputs $n_{max} \cdot (C+1)$ values. This also means that for each document n_{max} decisions are made per document, while n_i^* might be (a lot) smaller than n_{max}. That means that the model would output values for pages which do not exist. A solution to this problem could be to use white images as padding for the graphical data and $\mathbf{0}$ vectors as padding for the textual (vectorized) data, for x_i^j and y_i^j where $n_i^* \leq n_{max}$. The current problem formulation has three problems, namely,

1. Firstly, very large and sparse inputs and outputs are now used and produced. Since most of the documents would have fewer pages than the maximum, a lot of aforementioned padding is required. This increases the size of inputs, which require more resources in terms of computation.
2. Secondly, the input and output is of fixed size causing a constraint on the model's generalisability, i.e. if a sample outside $\mathcal{D}^*$ contains more than n_{max} pages the model could not process the sample as done in its training phase.
3. Thirdly, it is very much possible that some positional bias are in the data, meaning that some pages of certain document types are more likely to be on specific page indices. This bias is not wanted since it could be possible that in another (unknown) data set the positional distributions of the pages are different, which can result in faulty decisions by the model. This can also be seen as a generalisation problem.

In the next paragraph, the problem is formulated differently to tackle the mentioned problems.

3.1.2 Formulation II

As previously mentioned, the problem is reformulated to tackle input/output sparsity, generalisability and positional bias. First, the intermediate data set, $\mathcal{D}^{**}$ set is shown in Figure 3.1 (b). This data set contains the same data as $\mathcal{D}^{*}$, except now each file contains only one document type. So, each sub-document of the original files in $\mathcal{D}^{*}$ is now a separate instance. Next, a new synthetic data set, $\mathcal{D}_{synth}$, is made by (pseudo)randomly merging documents of $\mathcal{D}^{*}$. This is done to counter the generalisability and positional bias problems.

(a) $\mathcal{D}^{*}$ (original data set)

(b) $\mathcal{D}^{**}$ (intermediate data set)

(c) $\mathcal{D}_{synth}$ (synthetically merg

(d) $\mathcal{D}_{synth}^{sliced}$ (sliced synthetically merged data set)

Figure 3.1: Visualisations of $\mathcal{D}^{*}$, $\mathcal{D}^{**}$, $\mathcal{D}_{synth}$ and $\mathcal{D}_{synth}^{sliced}$. Each icon represents a page (textual and graphical) and its colour a document type.

The following data and variables are introduced,

$$\begin{aligned}
\mathbf{X}_i^{synth} &:= \text{ the graphical representations of the pages of document } i \\
\mathbf{Y}_i^{synth} &:= \text{ the textual representations of the pages of document } i \\
\mathbf{T}_i^{synth} &:= \text{ the document type of document } i \\
&\quad \text{with } j \in \{1, ..., n_i^{synth}\}, i \in \{1, 2, 3, ..., N^{synth}\} \\
n_i^{synth} &:= \text{ the number of pages of document } i \\
N^{synth} &:= \text{ the total number of documents after the original documents are} \\
&\quad \text{indexed per document type} \\
\mathcal{D}^{synth} &= \{(\mathbf{X}_i^{synth}, \mathbf{Y}_i^{synth}, \mathbf{T}_i^{synth}) \; \forall \; i \in \{1, 2, 3, ..., N_{synth}\}
\end{aligned}$$

By slicing the instances of $\mathcal{D}^{synth}$ with a window, a new data set, $\mathcal{D}_{sliced}^{synth}$, is formed. A few new parameters are defined for $\mathcal{D}_{sliced}^{synth}$, namely,

$$\begin{aligned}
k &:= \text{ the index of the page after which the model decides for a split} \\
m_1 &:= \text{ the number of pages before, and including, page } k \\
m_2 &:= \text{ the number of pages after page } k \\
window &= m_1 + m_2
\end{aligned}$$

Since it is known which documents were merged together, the page types and split points in the synthetic files are automatically annotated during the slicing procedure. Training a model on the samples of $\mathcal{D}_{sliced}^{synth}$ allows it to parse documents of any length, since each new document can be sliced into chunks and fed into the model. This lowers the model complexity compared to the first approach. The model only decides after page k if a certain document type ends or if no split is needed. To know which sub-document ends after a split decision, the model is trained to classify page k. This transforms the problem from a multiclass-multilabel problem to a binary classification problem (i.e. split or no split between page k and $k+1$) and a multiclass classification problem (i.e. classifying page k). To summarise, a file is sliced into chunks which are fed into the model, that outputs if a split is needed and what page type page k is, for each given chunk. After all the chunks of a file are processed, one has all the split decisions between consecutive pages and the predicted page types for every page in that file. The number of padding pages is reduced (compared to the first approach), since a window rolls over the document instead of giving the model the whole document. This means that there is no need to pad every document to a fixed length, reducing the sparsity of the input and output.

Assumptions

The described modelling approach (II) has a few assumptions, namely,

- Firstly, to train the model a data set is created using the following steps: 1) a data set is created by randomly merging documents from $\mathcal{D}^{**}$, 2) sliding over these samples using a window of size $m_1 + m_2$, which generates $\mathcal{D}^{synth}_{sliced}$. Since one knows which documents are merged, the annotation is done automatically. This modelling approach assumes that only nearby context and information, i.e. pages which are close to the split decision point, are sufficient to make a decision.
- Secondly, it is assumed that positional bias is reduced because of randomly merging documents. Thus, it is assumed that the model does not correlate a page index with a page type.

For this problem there are two direct optimised objectives and two indirect optimised objectives, discussed in the next paragraphs.

Direct optimised objectives

As explained in the modelling approach (II), there are two types of decisions the model makes, namely, the split decisions and the page classifications. The neural network will be optimised by directly minimising the *categorical crossentropy loss function* for the page classifications,

$$\begin{aligned} \text{loss}_{class} &= -\sum_{i=1} y_i \cdot log(\hat{y}_i) \\ y_{ij}, \hat{y}_{ij} &\in \{1, ..., C\} \end{aligned} \tag{3.1}$$

Next, the network will also be optimised by minimising the *binary crossentropy loss function* for the splitting part,

$$\begin{aligned} \text{loss}_{split} &= -\sum_{i=1} y_i \cdot log(\hat{y}_i) \\ y_{ij}, \hat{y}_{ij} &\in \{0, 1\} \end{aligned} \tag{3.2}$$

The total loss is a weighted sum of both losses,

$$\text{loss}_{total} = \beta_1 \cdot \text{loss}_{class} + \beta_2 \cdot \text{loss}_{split} \tag{3.3}$$

This loss is used by the backpropagation algorithm or optimiser (i.e. AdamW [44]) to lower the errors.

Primary indirect objective

The primary indirectly optimised objective is splitting a file into unnamed sub-documents. This objective is quantified by determining the proportion of the correctly made split or no split decisions by the model per file,

$$\mathrm{o}_{\mathrm{doc}}^{unnamed} = \frac{1}{N}\sum_{i=1}\frac{1}{n_i - 1}\sum_{j=1}^{n_i-1} I(y_{ij}, \hat{y}_{ij}) \tag{3.4}$$

$$I(a,b) = \begin{cases} 0, \text{if } a \neq b \\ 1, \text{if } a = b \end{cases}$$

$$y_{ij}, \hat{y}_{ij} \in \{0, 1\}$$

Each file needs n_i - 1 splitting decisions, since no decision needs to be made after the last page, hence the minus 1.

Secondary indirect objective

The secondary indirectly optimised objective by the model are the page classifications per file,

$$\mathrm{o}_{\mathrm{doc}}^{named} = \frac{1}{N}\sum_{i=1}\frac{1}{n_i}\sum_{j=1} I(y_{ij}, \hat{y}_{ij}) \tag{3.5}$$

$$I(a,b) = \begin{cases} 0, \text{if } a \neq b \\ 1, \text{if } a = b \end{cases}$$

$$y_{ij}, \hat{y}_{ij} \in \{1, ..., C\}$$

3.2 Baseline method: Support Vector Machine

As a baseline method, the Support Vector Machine (SVM) algorithm is used to classify file pages based on textual unigram features. This is also done by the authors of [61], except they also extract page topic features and used information of preceding pages. The Term Frequency - Inverse Document Frequency (TF-IDF) is used to represent the unigram features of a page as a real-valued vector. Based on the page classifications, the splits are inferred where each consecutive predicted page type ends.

3.3 Proposed methods

As discussed earlier, there are two directly optimised objectives, the total loss is quantified by Formula 3.3. To compute the losses, Formula 3.1 and 3.2, the model needs to output

two groups of prediction values, namely, the page classification and the split classification values. The proposed network architecture to compute these prediction values is shown in Figure 3.2. As one can see, there are two flows within this architecture, namely, the page classification and the split classification. The feature extractor converts the graphical and/or textual data into abstract features, which are then used to compute the two groups of prediction values.

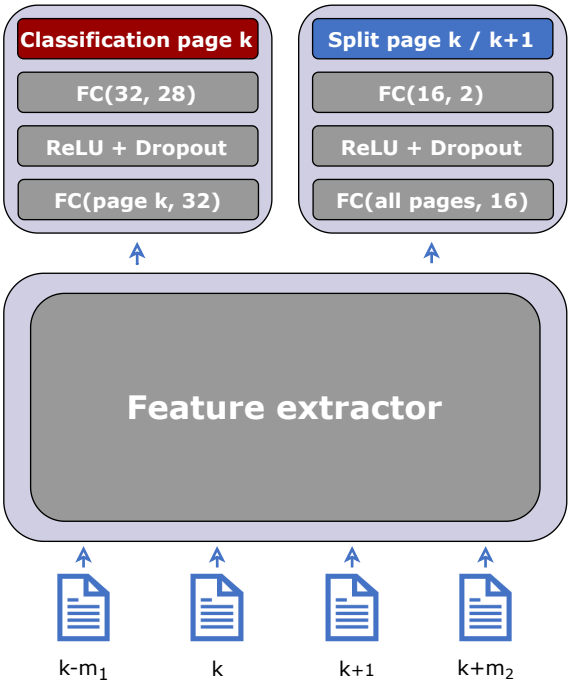


Figure 3.2: A diagram of the general architecture for classifying pages and splitting documents.

Both classification networks contain two fully connected layers and a Rectified Linear Unit (ReLU). At last, both networks compute the predictions using the softmax function. During the trial runs, several other classification networks are tried (e.g. more/fewer layers, more/fewer nodes, tanh activation function) but the proposed networks seem to be the most stable and best performing during training. In the next paragraphs, three types of feature extractors are discussed, namely, the Time Distributed Convolutional Neural Network (TDCNN), the Recurrent Neural Network (RNN) and a combination of both.

3.3.1 Time Distributed Convolutional Neural Network

From the literature discussed in Chapter 2, a lot of image based features are handmade or extracted by a separate algorithm [3, 45]. There is published work by [61], where two net-

works are trained separately (as feature extractors) and used by a Multilayer Perceptron (MLP) classifier. The main interest in this thesis is to let the model decide which parts of the input are used to contribute to the decision-making, instead of (semi-)manually engineering features. To automatically extract (abstract) features from a number of consecutive images, the Time Distributed Convolutional Neural Network (TDCNN) is used. Since one of the goals is to make a decision if a split is needed for a given input, multiple pages are needed as input for the model. Each page, represented as an image, goes through a CNN and is then represented by a vector (of fixed size). The time distributed part of this feature extractor means that each page of an input is represented by the same CNN. Say one uses four input images to decide if a split is needed between page 2 and 3, there does not seem to be any reason why each page should be represented by a different feature extractor (CNN). The assumption behind this method is that extracting features from a page should not be conditioned on the place of a page in the input order. This time distributed part is also done to limit the number of parameters in a model, i.e. parameter sharing between page images.

The feature extractor consists of several stacked TDCNN blocks. Each TDCNN block has a convolutional layer with a predefined kernel size and number of filters, a max pooling layer with a predefined kernel size and a ReLU activation function. From the early trials, it is observed that max pooling performs better compared to average pooling. Also, the tanh activation function is used in the early trials, the results of the tanh activation function do not differ much compared to ReLU activation, but the networks trained with the ReLU activations seem to be more stable during training. Thus, the ReLU activation function is chosen. Next, the representations of all the pages of an input are concatenated and are then used as input for the split classification network. The representation of page k is used as input for the page classification network. This is shown in Figure 3.3.

A note on LayoutLMv2

In extension to Chapter 2, the LayoutLMv2 (and LayoutXLM) model uses a visual encoder, with ResNeXt-FPN [62] as a backbone, and is trained a large document data set. During this research, the pretrained visual encoder of the LayoutLMv2 model is used as a (time distributed) feature extractor for the page images. Since the visual encoder has seen a lot of documents, it is assumed that it could also work as a feature extractor for this particular problem. From the trial runs, it seems that the classification networks could not effectively learn how to use the representations given by the pretrained visual encoder.

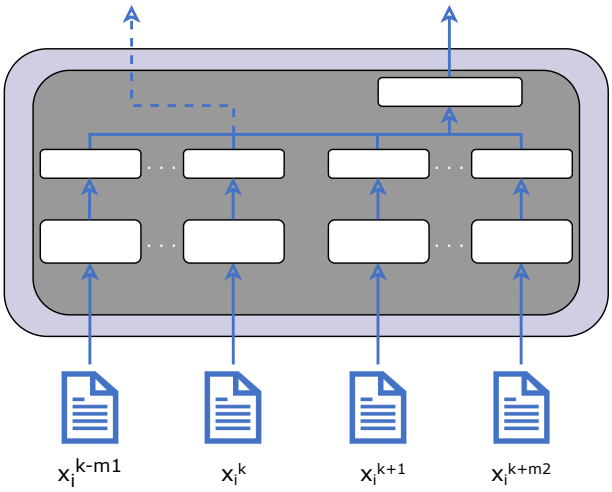


Figure 3.3: The TDCNN feature extractor for page images. Each page image is represented by the same TDCNN feature extractor. All of the final TDCNN outputs are used by the split classification network. Only the TDCNN output of page k is used for the page classification network (dotted line).

3.3.2 Recurrent Neural Network

To extract features from the textual content of a page, a Recurrent Neural Network is used to process text. Each page is represented by a fixed sized vector of size H. Several types of recurrent units can be used as feature extractors, as shown in Figure 2.2. From the trial runs, all recurrent units perform well on the data. The Elman Recurrent Unit is chosen since it is less complex compared to the other recurrent units. As one can see in Figure 3.4, the first hidden vector, h_0, is initialised as a $\mathbf{0}$ vector. Next, each time that the textual content of a page is processed by the unit, the hidden vectors are given as output. Furthermore, the hidden output of all the pages are concatenated and used by the split classification network. Also, the hidden output of page k is used as input for the page classification network.

3.3.3 Hybrid

The hybrid feature extractor is a hybrid between the TDCNN and RNN feature extractors. As shown in Figure 3.5, the page images and textual content are processed by the TDCNN and Elman RNN, respectively. Next, the TDCNN and RNN outputs are all concatenated and are used as input for the split classification network. The TDCNN and RNN representations of page k are concatenated and used as input for the page classification network.

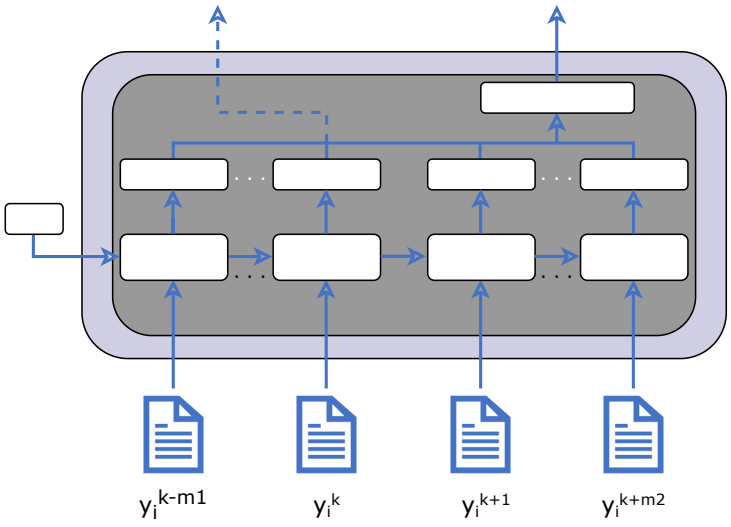


Figure 3.4: The Elman RNN feature extractor for textural data. The split classification network uses all the hidden outputs and the page classification network only uses the hidden outputs of page k (dotted line).

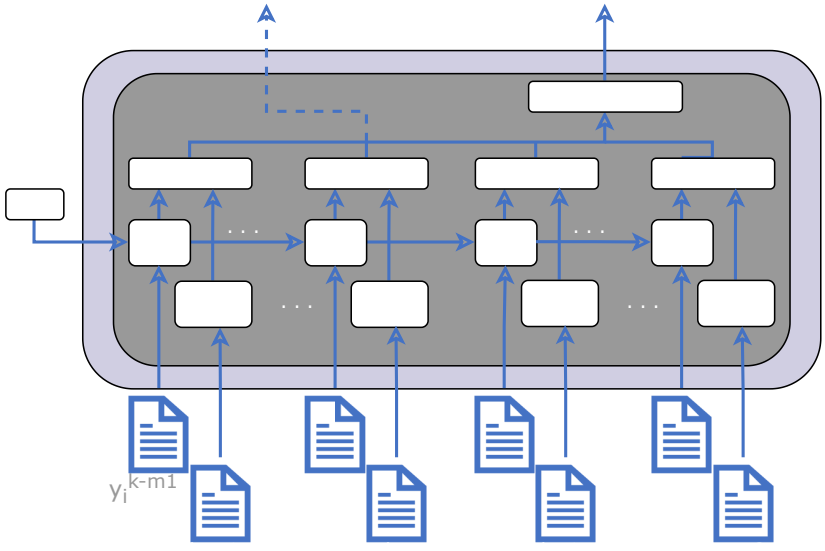


Figure 3.5: The hybrid feature extractor contains a TDCNN and RNN feature extractor to process images and textual content, respectively. The split classification network uses all the hidden outputs and the page classification network only uses the hidden outputs of page k (dotted line).

3.4 Evaluation methods

To understand how the models are performing, they are evaluated using quantitative and qualitative methods.

3.4.1 Quantitative evaluation

To evaluate the models in quantitative way, the previously mentioned indirect objective Formulas (3.4) and (3.5) are used as metrics. Formula (3.4) can be interpreted as the mean percentage of correct splitting decisions per file. Formula (3.5) can be interpreted as the mean percentage of correct page classifications per file. In extension to these objectives, a metric is used to see if the model correctly identifies whole sub-documents in a file,

$$\mathrm{o}_{\mathrm{doc}}^{\textit{sub-document}} = \frac{1}{N}\sum_{i=1}\frac{1}{n_i^{sub}}\sum_{j=1}\mathbb{1}_{\text{identified sub-document j}} \tag{3.6}$$

$$n_i^{sub} := \text{number of sub-documents in file i}$$

The definition of a fully correctly identified sub-document consist of two parts, namely, a sub-document contains the expected number of pages and is classified as its expected document type. Since the page types are known, the expected number of pages and sub-document type can be inferred. Since the SVM uses page type predictions to indirectly infer the file splits, a sub-document cannot be ambiguous (it is either correctly identified or not). The neural networks do not infer the file splits from the page classifications but directly use the input to make the splitting decisions. Given that a sub-document is correctly extracted (using the split classification network), one can use the page classification network to predict the sub-document type. Since the splits are already decided, one can use a threshold value (e.g. 50%) to decide which document type the extracted sub-document is, based on the page type predictions of that sub-document. In contrast to the SVM, if a page classification is wrong, it is still possible to infer the right sub-document type. This means that the length of a sub-document is not important in (3.6), either the whole sub-document is correctly identified or not.

3.4.2 Qualitative evaluation

To give insight why the model decides to split between page k and k+1 and classify page k with a certain page type, the Gradient-weighted Class Activation Mapping (Grad-CAM) [52] is used to display the importance of certain parts of the visual input used by a CNN. Grad-CAM uses the target(s) and gradients to produce a heatmap which is laid over the input images. A customized implementation of [22] is used.

4

Optical Character Recognition

In this Chapter several OCR engines, and their performance, are discussed. In this thesis two modalities, the textual and graphical data, are used. To get the graphical data from a PDF page, one simply converts the page to pixels using software. To extract the text from a PDF page, a text extraction tool is used. Although this text extraction tooling is sufficient for most PDFs, a (small) portion cannot be read due to text encoding/decoding issues or a document contains scanned pages. For these cases, one can use OCR models to extract text from a page. In the next paragraphs, off-the-shelf OCR models are tested to see which are suitable for the data used in this thesis. As explained in Chapter 2, an end-to-end OCR application consist of text detection and text recognition components [18, 33, 38, 43]. Text detection models isolate and extract the part(s) of an image which (probably) contain text. Text recognition models use those extracted parts to convert the image to a character sequence. i.e. words/sentences.

OCR methods

To test which OCR method is suitable for the given data, a few requirements are set,

- The OCR method is already implemented, usable and practical. To implement a novel OCR method from scratch is out-of-scope during this project.
- The OCR engine is capable of extracting text based on the Latin alphabet (Chinese and Arabic symbols are not a requirement).

The following combinations of text detection and recognition models are used for the benchmark, implemented by [47],

- Text Detection: Differentiable Binarization (DB) with Resnet50 and DB with MobileNetV3-Large [32, 39, 47]
- Text Recognition: CRNN with VGG16, CRNN with MobileNetV3-Small and CRNN with MobileNetV3-Large [32, 53, 56]

Apart from these combinations of deep learning models, the Tesseract Engine (v5.0.0) [58] is used. This OCR engine is an open source end-to-end OCR application developed originally by HP and sponsored by Google. This is also the OCR engine used by [34, 63, 64, 65] and can be considered as a practical industry standard open source OCR tool.

Benchmark data sets

Two public benchmark data sets are used to test the different models, namely, the FUNSD [36] and CORD [49] data set.

Form Understanding in Noisy Scanned Documents (FUNSD)

This data set, by [36], contains in total 199 scanned documents such as letters, e-mails, magazines and other forms from the 1980s-1990s. This data set is a subset from the Ryerson Vision Lab Complex Document Information Processing (RVL-CDIP) [28]. The documents are 1-channel grayscalled images, each page has text and bounding boxes annotations.

Consolidated Receipt Dataset (CORD)

This data set, by [49], contains in total 11,000 scanned receipts. The receipts are 3-channel RGB images originating from Indonesia and gathered through crowd-sourcing. These images also have text and bounding boxes annotations. In this thesis, the images of the first release of the training data set (800 images) and test data set (100 images) are used to evaluate the OCR models.

Metrics

To evaluate and compare the methods in a quantified manner, the recall, precision, Intersection-over-Union (IoU) and computation time are used.
As explained, OCR has two main modules: detecting where the text is and what type of characters are in the found text areas. The outcome of both parts are integrated into the

recall and precision. To understand how these metrics are computed, the definitions, based on the documentation of [47], are given below,

$$
\begin{aligned}
\mathcal{B} :=&\ \text{the set of possible bounding boxes} \\
\mathcal{L} :=&\ \text{the set of possible character sequences} \\
S :=&\ \text{the total number of images in the set} \\
G_k :=&\ \text{the number of ground truths of image k} \in \{1, 2, ..., S\} \\
P_k :=&\ \text{the number of predictions of image k} \in \{1, 2, ..., S\} \\
t :=&\ \text{the } IoU \text{ threshold for accepting the assignation of predicted} \\
&\ \text{bounding box and ground truth bounding box} \\
G_k, P_k \in&\ \mathbf{Z}^+ \\
t \in&\ [0, 1]
\end{aligned}
$$

Given the above values and sets, the metrics per image are defined as follows,

$$
\begin{aligned}
\forall (B, L) \in \mathcal{B}^{G_k} \times \mathcal{L}^{G_k} \\
\forall (\hat{B}, \hat{L}) \in \mathcal{B}^{P_k} \times \mathcal{L}^{P_k}
\end{aligned}
$$

$$
\begin{aligned}
Recall_k : R(B, L, \hat{B}, \hat{L})_k &= \frac{1}{G} \sum_{i=1}^{G} h_{B,L}(\hat{B}_i, \hat{L}_i) \\
Precision_k : P(B, L, \hat{B}, \hat{L})_k &= \frac{1}{P} \sum_{i=1}^{P} h_{B,L}(\hat{B}_i, \hat{L}_i) \\
Intersection\ over\ Union : IoU(A, B) &= \frac{A \cap B}{A \cup B} \\
\forall (b, l) \in \mathcal{B} \times \mathcal{L}\ , h_{B,L}(b, l) &= \begin{cases} 1, \text{if } b \text{ has been assigned to a given } B_j \\ \quad \text{with an } IoU \geq t \text{ and that } l = L_j \\ 0, \text{ otherwise.} \end{cases} \\
k &\in \{1, 2, ..., S\}
\end{aligned}
$$

To evaluate the whole set, one simply takes the mean of the metrics over all images,

$$
\begin{aligned}
Recall : R &= \frac{1}{S} \sum_{k=1}^{S} R_k \\
Precision : P &= \frac{1}{S} \sum_{k=1}^{S} P_k
\end{aligned}
$$

In words, for each image k a golden set of bounding boxes ($\mathcal{B}$) with each a character sequence ($\mathcal{L}$), i.e. text, is given. The OCR model also returns all the predicted bounding boxes and their characters sequences. To check if the model found the right text, the golden and predicted bounding boxes need to be matched. Since an exact match of the bounding boxes is nearly impossible, the IoU is used as a match indicator. The IoU is a metric commonly used in the *object detection* field [48]. If the IoU of a pair golden and predicted bounding boxes is larger than the threshold value t, than it is assumed that the pair is a match. Next, the text of the matched bounding boxes need to be the same, i.e. $l = L_j$. In this thesis, a strict character matching method is used, one could also use a more loose character matching method. If these two requirements are met, the indicator function ($h_{B,L}$) will return 1, otherwise 0. For each image k, the indicator function can be used to determine the *recall* and *precision* of the OCR engine. To calculate the recall and precision over the whole set, one simply takes the mean based on the metric values per image k.

Results OCR methods

The results on the benchmark marks data sets, FUNSD and CORD, are shown in Table 4.1. As one can see, there is a difference in performance between the models. Based on results of the model combinations in Table 4.1, it seems that the ResNet50 model outperforms the MobileNetV3 (Large) model on the CORD data set, the performance on the FUNSD data set is almost the same. Next, the performance or the recognition models are almost the same, there are only some minor differences. A small note on the Tesseract OCR engine, it performs quite poorly on the benchmark, but it is still an interesting option since it runs on a CPU device, while the other models require a GPU device (for an acceptable inference time). To conclude, the ResNet50 model is chosen as the text detection model. The CRNN MobileNetV3 (Small) is chosen since the performance of the recognition models are almost the same, except that this model has the least parameters.

Table 4.1: The mean (and standard deviation) of the precision, recall and computation time (CT) of the OCR engines on the FUNSD and CORD benchmarking data sets. The precision and recall are presented in percentages and CT is in seconds. To compute the metrics, the IoU threshold (t) is set to 0.5 and all characters, including the case type, need to be matched by the models to set $h_{B,L}(b,l)$ to 1. All the models, except the Tesseract engine, are used with GPU acceleration (NVIDIA QUADRO RTX 5000), the computation time is much higher for these models if inference is done on a CPU device.

		CORD			**FUNSD**			
Detection	**Recognition**	**Recall**	**Precision**	**CT**	**Recall**	**Precision**	**CT**	**# Parameters**
DB ResNet50	CRNN VGG16	**85.07** ± 14.35	**83.33** ± 15.54	0.56 ± 0.07	66.65 -12.74	74.38 ± 10.54	0.58 ± 0.04	41M
	CRNN MobileNetV3 (Small)	84.36 ± 14.26	82.63 ± 15.44	0.55 ±0.07	67.25 ±12.74	75.02 ± 10.26	0.55 ± 0.03	27.3M
	CRNN MobileNetV3 (Large)	85.02 ± 14.54	83.29 ± 15.75	0.55 ± 0.07	67.77 ±12.82	**75.60** ± 10.41	0.55 ± 0.03	29.7M
DB MobileNetV3 (Large)	CRNN VGG16	68.89 ±25.81	59.47 ±26.98	0.54 ± 0.08	66.78 ±13.17	71.28 ±12.16	0.57 ± 0.05	20M
	CRNN MobileNetV3 (Small)	68.30 ±25.50	58.97 ±26.73	0.54 ± 0.08	67.56 ±12.84	72.11 ± 11.67	0.54 ± 0.03	6.3M
	CRNN MobileNetV3 (Large)	69.03 ± 25.68	59.60 ± 26.88	0.54 ± 0.08	**67.86** ±13.25	72.40 ± 12.05	0.54 ± 0.03	8.7M
Tesseract (English)	Tesseract (English)	35.07 ± 27.82	18.93 ± 17.14	1.23 ± 1.27	44.48 ±15.84	47.60 ± 14.93	0.92 ± 0.16	Unknown
CRAFT	VGG + Bi-LSTM	27.65 ± 17.10	34.72 ± 20.30	0.44 ± 0.28	15.31 ±11.11	27.49 ± 14.03	1.19 ± 0.53	Unknown

5

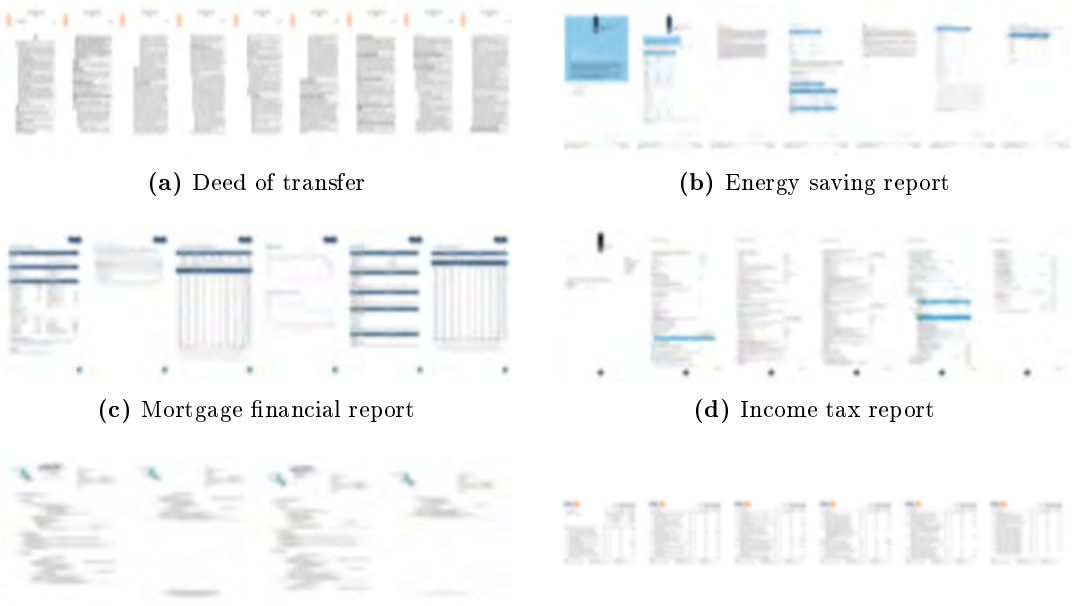
Data

The data ($\mathcal{D}^*$), as provided by the organisation, consists in total of 32,142 PDF files. These files are originally annotated by a rule based algorithm which looks at particular key-words to classify the document type. By sampling random files and visually inspecting them, it can be said that this algorithm does not perform well. This algorithm has two main problems, namely,

1. Due to the key-word based approach, ambiguous annotations are given to files where the text could not be (entirely) extracted. This is expected since no OCR software was originally used to get the text from the PDF, only a PDF text extractor was used to get the text. The PDF reader might be sufficient for most digital files, such as auto-generated files or standardised forms, but files can contain scanned pages or images which cannot be read by a PDF text extractor.

2. The algorithm assumes that each file can only contain one document type, this is not true for files in $\mathcal{D}^*$. This also causes ambiguous classifications.

Since these problems could significantly affect the performance of a model, the data is annotated by hand. The annotation is done using self-made annotation software which converted each page of a file into an image and asked the annotator what class the page belonged to. The document types are provided by sources from within the company. A distilled and customised version of the original list is used to annotate the pages since a lot of the documents could be grouped into a more general class or could not be directly mapped to a category in the list. In total, 28 document types are used in the annotation scheme, including a rest category. Since each file is page-wise annotated, all the original files are split (if applicable) and categorised into their document class. This can be seen as the intermediate data set ($\mathcal{D}^{**}$), a few examples of these documents are shown in Figure



(a) Deed of transfer

(b) Energy saving report

(c) Mortgage financial report

(d) Income tax report

(e) Property information

(f) Bank account statement

Figure 5.1: Multi-page document examples of six document types. All shown examples are cropped and blurred since the data is personal and private.

5.1. During the annotation phase it seemed that there were two types of files, namely, files which only contain one document type and files which contain more than one document type. To leverage the information of the files containing only one document type and to generalise the problem and solution methods, the decision is made to synthetically generate (pseudo-)randomly data sets using the document files in $\mathcal{D}^{**}$ (which is discussed in a later section). Furthermore, as one can see in Figure **??** (red bars), the page type distribution is heavily imbalanced. Next, to remove any double instances, each page was converted to an image and was encoded to a character sequence using the Secure Hashing Algorithm 1 (SHA-1) [19]. If an exact character sequence match was found, the first instance was kept and any other instances with that particular encoding were removed. The page type distribution after the duplicates were removed can be seen in Figure **??** (blue bars). As one can see, some documents types contain a lot of double instances. Also, the mean number of pages for each document type are wildly different, as shown in Figure **??**.

5.1 Sets

The intermediate data set is split into three parts: the train, validation and test set. The train set contains 60% of the data, the validation and test set each contain 20%. The split

is done in a stratified way. This split is made on the intermediate data set since no data leakage, i.e. (parts of) documents are not in multiple sets, between sets is possible. Since the aim of this research is to split files into logical sub-documents, the documents in each set are merged with a merging method. This merging method generates synthetic data sets ($\mathcal{D}_{synth}$), for training, validating and testing.

5.2 Processing data

To use the graphical and textual data for the model and/or OCR engine, one needs to prepare and process the data. In the next few paragraphs the data processing is discussed.

5.2.1 Model image processing

As explained, there seems to be an imbalance in the number of documents between document types. To increase the number of samples for the smaller document types, data augmentation methods are used. Data augmentation methods are methods to increase the sample size and quality of a (train) data set [55]. There are numerous methods which can be used for data augmentation, such as General Adversarial Networks (GANs) for generating new training instances. It is shown in other fields (e.g. medical [8], public data sets [67]) the model performance does increase when using this type of synthetic data. In the case of the current data, document pages, there is a lot of contrast between neighbouring pixels (i.e. black symbols and white background), the document is also used for the OCR engine which requires (Dutch) visible placed words on the document and the document pages need to resemble each other in terms of textual and graphical content. Given these data characteristics and current accessible technology, it is out of scope to use these GAN type of methods for document generation, although it is possible to generate binary images to some extent [9]. What can be used are the following, more traditional, augmentation methods from [55],

- *Colour space*
  Each graphical representation of page j of document i (x_i^{*j}) is a tensor of (height $\times$ width $\times$ colour channels). A page can be augmented by randomly changing the values in the colour channels.

- *Resizing*
  A page can be resized with different scale values, e.g. 0.8, 0.5, etc., centered and padded with white pixels to the required input shape of the model.

- *Rotation*
  Document pages in the provided data set ($\mathcal{D}$) are sometimes scanned, and thus a bit rotated. So, rotating pages is a natural augmentation choice for this type of data, although the rotation will be at max 5° to the right or left.

- *Translation*
  A document page can be shifted left, right, up and down with a random amount of pixels. The limit is set to 150 pixels.

- *Noise injection*
  Random noise is added to the document page. Two types of noise are used, namely, salt and pepper noise or Gaussian noise.

The number of augmentations per newly generated instance is randomly chosen from a Uniform(1,3) distribution. Since each augmentation cannot be read by a PDF reader but needs to be processed by an OCR engine, the number of augmentations is limited due to computational and storage resources. The minimal number of pages per document type is set to 500 and based on this threshold the number of copies per document type is calculated. The augmentations are only done on the documents in the training data set. Each page is resized to a width of 60 pixels and a height of 90 pixels and has three channels (RGB). Next, each pixel value is divided by the maximum pixel value, 255, to normalise the input.

5.2.2 OCR image processing

If the reader cannot read more than 30 words (i.e. separate character sequences divided by a whitespace) of a page, the page is flagged and will go through the OCR engine. The (two stage) OCR engine is chosen based on the results of the benchmarks in Chapter 4. The DB ResNet50 model is chosen for the text detection stage and the CRNN-MobileNetV3 (Small) is chosen for the text recognition stage. This combination is chosen since it performed quite well on the benchmark data sets in general. In total, including the augmented pages in the training data, 62,031 out of 154,675 pages (roughly 40%) require the OCR engine.

5.2.3 Processing textual data

The text processing is simple and straightforward. A tokenizer with a vocabulary of 1,500 words together with a Term Frequency - Inverse Document Frequency (TF-IDF) vectorizer

is fitted on the training data. Thus, the textual content of each page is converted to a vector with a length of 1,500.

5.3 Merging documents

By (pseudo)randomly merging documents, multi-document files are generated. Next, a window of size m_1 + m_2 slides over the files making samples which can be used as input for the model. Since the document types and their place inside the files are known, the page types and sub-documents endings are automatically annotated during the sliding process. As can be seen in Figure **??**, there is a heavy imbalance in the number of documents per document type. It is assumed that if a pure random merging method is used, i.e. randomly select documents and merge them into one file, this imbalance will also be in the multi-document files and can influence the generalisability of the model. To make sure the influence of the imbalance is limited during optimisation, a conditional merging procedure is developed.

5.3.1 Model data

The procedure takes a random document of each document type and merges it in a random order, this is a sample of $\mathcal{D}_{synth}$, i.e. $(\mathbf{X}_i^{synth}, \mathbf{Y}_i^{synth}, \mathbf{T}_i^{synth})$. So, a sample does not contain multiple documents of the same document type. Each document which is picked is not replaced, unless there are no documents left of that document type. In this case the whole set of that particular document type is added to the data set again. This means that document types which are relatively large are being undersampled, while document types which are relatively small are being oversampled (since they are replenished when empty). Using this approach, another data imbalance problem occurs, namely, the number of 'No split' labels is much larger than the number of the 'Split' labels. This is because multipage files need less splits since the number of pages is larger than the number of sub-documents in that file. To counter this, 10% of the total 'No split' labels is randomly kept during this procedure. At last, since it is hard to generate the same number of samples for each document type, only a lower bound can be set on the number of samples (since each document type is selected per sample). Although the document imbalance is reduced, there is still an imbalance in the new data set, $\mathcal{D}_{sliced}^{synth}$. This is because document types which have more pages per document are now (slightly) being overrepresented.

5.3.2 Evaluation data

To evaluate a model properly, one is mainly interested on how a model performs on multi-document files, and not just chunks of those files. To make those multi-document files, one needs to sample documents and merge them. Ten documents are randomly chosen (each with a different document type) and randomly merged together, without replacement. This is done a hundred times, if a document type is not available any more in the set, it is not replenished, otherwise a document would be evaluated multiple times and bias the score. This is called an evaluation set. Since it is possible that one could cherry-pick documents which work well with the model and/or the randomness generates a particular good or bad evaluation set, this is done again a hundred times. Thus, the resulting scores are first averaged per evaluation set, and then again averaged over all evaluation sets. This is done to give a more general insight into the performance of the model.

6

Experiments

In this chapter the model architectures, experimental details and results are discussed.

6.1 Architectures

In the next paragraphs the SVM hyperparameter settings and several feature extractors for the network are discussed.

6.1.1 Support Vector Machine

The SVM model is used as a baseline by classifying each page of a document separately. The SVM uses a linear kernel, C is set to 1, L2 penalisation is used and is optimised using the squared hinge loss. The model is trained on the training data set without the synthetic data samples.

6.1.2 Time Distributed Convolutional Neural Network

The TDCNN feature extractor consists of four stacked TDCNN blocks. As explained in Chapter 3, each block consists of a convolutional layer, a max pooling layer and ReLU activation function. The kernel size of the convolutional and max pooling layers are set on (2,2). The number of filters in the TDCNN blocks are set to 30, 60, 90 and 120. From the trial runs, adding more layers and/or filters does not give a better result. This feature extractor maps each input image to a vector of size 360. Three models containing this feature extractor are trained and tested, namely, TDCNN$^{(1:1)}$, TDCNN$^{(2:2)}$ and TDCNN$^{(3:3)}$. The number of front and back context pages are denoted as (m_1:m_2). The number of trained parameters are approximately 187k, 200k and 211k, respectively.

6.1.3 Recurrent Neural Network

The textual data of each page is vectorized to a vector of size 1,500, using the TF-IDF vectorization method. The Recurrent Neural Network has a (small) hidden layer of size 75 and uses the ReLU as a non-linearity. From the trial runs, it seems that compressing the data with one hidden layer containing 75 nodes works well and limits the number of parameters in the model. Thus, the textual content of each page is represented by a vector of size 75. This feature extractor is used by three models, namely, $\text{RNN}^{(1:1)}$, $\text{RNN}^{(2:2)}$ and $\text{RNN}^{(3:3)}$. The number of trainable parameters are approximately 125k, 127k and 129k, respectively.

6.1.4 Hybrid

The hybrid model uses the TDCNN and RNN feature extractors. The textual and graphical content of each page are represented by a vector of size (360+75=)435. The three models which are trained and tested are TDCNN-$\text{RNN}^{(1:1)}$, TDCNN-$\text{RNN}^{(2:2)}$ and TDCNN-$\text{RNN}^{(3:3)}$, containing 311k, 325k and 339k parameters, respectively.

6.1.5 Experimental details

To train the discussed models some hyperparameters are estimated, this is done using trial runs. The following hyperparameters, and their values, are used:

- The *learning rate* is set to 1e-3.
- The *batch size* is set to different values, depending on the architecture of the model. The used batch size per trained model is mentioned in Appendix A.
- The *weight decay rate* is set to 1e-2.
- The number of *epochs* is set to different values. Each model is trained five times for six epochs, as shown in Appendix A. Based on the divergence between the training and validation losses, the number of epochs is chosen per model.
- The *dense dropout rate* is set to 5e-1. This rate is used for the dense layers after the convolutions and/or recurrent units. This a relatively high dropout rate. From the trial runs, it seems that a lower rate makes the model overfit and a higher rate does not let the model learn properly.

- The *feature map dropout rate* (i.e. 2D-dropout) is set to 1e-1. This rate is used for the feature maps (i.e. channels) produced by the convolutional filters. From the trial runs, a lower rate makes the model overfit in some cases, a higher rate does let the model learn.

- The *number of context pages*, m_1 and m_2, are set to different values. When more context pages are added, the number of parameters in the model also increase. From the trial runs, it seems that adding more pages can cause to the model to overfit and training the models with more pages requires more resources (in terms of memory and storage). It is decided that $0 < m_1, m_2 \leq 3$ and that $m_1 = m_2$, because this limits the scope of the project, is in bounds of the available resources and there is no reason to assume that $m_1 \neq m_2$.

- The *vocabulary size*, H, is set to 1,500 words.

- The *width* of a page (in terms of pixels), w, is set to 60.

- The *height* of a page (in terms of pixels), h, is set to 90.

- The *number of channels* is set to 3. From the trial runs, it seems that binarization (i.e. 1 black and white channel) removes too much (colour) information to let the model learn properly.

- The *page classification output nodes* are set to 28, as discussed in Chapter 5.

- The loss weights, β_1 and β_2, are set to 1 and 3, respectively. During the trial runs, β_1 was kept at 1 and β_2 was tuned until a good value was found.

Furthermore, the results on the training and validation data set are made with the models which are only trained on the training data (including the augmented data). The results on test data set are made with the models which are trained on the training (including the augmented data) and validation data set.

6.2 Evaluation

The evaluation is done using quantitative and qualitative methods.

Table 6.1: The $o_{doc}^{unnamed}$ scores of the models on the training, validation and test data in terms of means and standard deviations (in %). The context (i.e. pages before and after the split decision), if applicable, is denoted as $(m_1 : m_2)$. The average and standard deviations are computed based on the means of the evaluation sets (100 in total).

Model / **Data set**	**Train**	**Validation**	**Test**
SVM	**98.81** ± 2.69	94.03 ± 9.09	93.06 ± 9.29
$RNN^{(1:1)}$	98.07 ± 3.12	**95.93** ± 5.29	95.13 ± 7.20
$TDCNN^{(1:1)}$	92.87 ± 3.25	90.45 ± 10.00	91.72 ± 8.32
$TDCNN\text{-}RNN^{(1:1)}$	97.73 ± 7.12	95.42 ± 5.32	95.95 ± 5.24
$RNN^{(2:2)}$	97.39 ± 3.55	95.08 ± 5.47	95.65 ± 5.22
$TDCNN^{(2:2)}$	93.41 ± 6.41	90.95 ± 9.38	94.20 ± 6.44
$TDCNN\text{-}RNN^{(2:2)}$	97.79 ± 3.06	94.62 ± 5.91	96.00 ± 4.77
$RNN^{(3:3)}$	95.98 ± 4.99	93.43 ± 8.34	95.01 ± 6.23
$TDCNN^{(3:3)}$	93.24 ± 6.38	90.57 ± 9.73	92.08 ± 8.08
$TDCNN\text{-}RNN^{(3:3)}$	97.10 ± 3.81	94.72 ± 5.64	**96.87** ± 3.80

6.2.1 Quantitative results

The results, in terms of the evaluation metrics mentioned in Chapter 6, of the models are shown in Table 6.1, 6.2 and 6.3. As one can see in Table 6.1, the SVM model is best with the splitting classification on the training data set, but the $RNN^{(1:1)}$ and $TDCNN\text{-}RNN^{(3:3)}$ are best at splitting on the validation and test data sets, respectively. Next, the standard deviations of the TDCNN models are in most cases relatively higher than that of the other models. This means that the splitting performance of this model is not consistent over all evaluation sets, compared to the other models.

Next, as can be seen in Table 6.2, the SVM is best at classifying correctly the pages in a file, on all the data sets. It is noticeable that the test score is sometimes higher than the validation and/or training score, this is probably because the test scores are made with the model trained on the training and validation data (instead of just the training set). Again, the TDCNN models have relatively large standard deviations. Thus, the model is not that consistent in its page classification performance over the evaluation sets. The recall and precision of the $o_{doc}^{unnamed}$ and o_{doc}^{named} metrics are shown in Appendix B.

At last, Table 6.3 shows the average and standard deviation of the mean percentage of fully extracted sub-documents in an evaluation set. Again, the SVM outperforms the other models on every data set. The standard deviations seem to be high for every model on all

Table 6.2: The o_{doc}^{named} scores of the models on the training, validation and test data in terms of means and standard deviations (in %). The context (i.e. pages before and after the split decision), if applicable, is denoted as $(m_1 : m_2)$. The average and standard deviations are computed based on the means of the evaluation sets (100 in total).

Model / **Data set**	**Train**	**Validation**	**Test**
SVM	**98.23** ± 3.76	**91.77** ± 12.01	**92.14** ± 9.92
$RNN^{(1:1)}$	91.46 ± 9.55	87.59 ± 13.40	90.19 ± 10.79
$TDCNN^{(1:1)}$	75.05 ± 17.75	68.36 ± 21.72	74.25 ± 20.05
$TDCNN\text{-}RNN^{(1:1)}$	92.70 ± 9.16	86.66 ± 14.54	90.12 ± 10.94
$RNN^{(2:2)}$	87.71 ± 11.62	84.91 ± 14.47	89.15 ± 12.37
$TDCNN^{(2:2)}$	80.05 ± 15.86	71.05 ± 21.11	73.97 ± 19.77
$TDCNN\text{-}RNN^{(2:2)}$	94.46 ± 7.80	88.71 ± 14.05	91.68 ± 9.94
$RNN^{(3:3)}$	91.11 ± 9.57	88.74 ± 13.17	91.53 ± 9.82
$TDCNN^{(3:3)}$	76.49 ± 17.00	68.98 ± 21.51	73.82 ± 20.33
$TDCNN\text{-}RNN^{(3:3)}$	88.30 ± 11.45	80.53 ± 16.94	91.29 ± 10.50

the data sets. This means that, in general, the models are not consistent in fully correctly extracting all the sub-documents in an evaluation set.

6.2.2 Qualitative results

To investigate if the models with an integrated vision part actually look at visually interesting areas of a page (e.g. recurring symbols/logos, text alignments, colourful areas, table layouts), the gradients and the output of each convolutional layer are used to make a heatmap, which is projected on the original input image. This is done with the Grad-CAM method and is shown in Figure 6.1 for the models that use $(m_1 = m_2 =)3$ context pages. As one can see, the first convolutional layer (a) highlights some parts of the textual areas and logos. From inspecting other samples, it seems that the models also pick up visually interesting areas in other types of documents. The heatmaps of the second convolutional layer (b) consist of relative small heated areas, which seem to be randomly distributed across the pages. Although some samples show horizontal heated lines, which seem not to be related to the layout of a page (e.g. the second row and first page of Figure 6.1 (b)). The third convolutional layer (c) outputs relatively larger heated areas, also these seem to be random distributed on the pages. At last, the heatmaps of the fourth and final convolutional layer (d) consists of somewhat smoother and large heated areas. These also seem to be non-related to visually interesting areas and some pages seem to have very few

Table 6.3: The $o_{\text{doc}}^{sub\text{-}document}$ scores of the models on the training, validation and test data in terms of means and standard deviations (in %). The context (i.e. pages before and after the split decision), if applicable, is denoted as $(m_1 : m_2)$. The average and standard deviations are computed based on the means of the evaluation sets (100 in total).

Model / **Data set**	**Train**	**Validation**	**Test**
SVM	**92.94** ± 8.10	**86.25** ± 12.11	**85.99** ± 11.88
RNN$^{(1:1)}$	87.40 ± 10.22	82.58 ± 13.36	84.33 ± 12.53
TDCNN$^{(1:1)}$	69.05 ± 13.91	67.08 ± 16.54	71.02 ± 15.69
TDCNN-RNN$^{(1:1)}$	87.51 ± 10.52	78.79 ± 15.22	84.16 ± 13.01
RNN$^{(2:2)}$	81.96 ± 12.17	77.10 ± 15.48	79.94 ± 14.55
TDCNN$^{(2:2)}$	74.18 ± 13.12	70.48 ± 15.58	73.19 ± 15.30
TDCNN-RNN$^{(2:2)}$	87.62 ± 10.27	78.61 ± 14.70	81.66 ± 14.38
RNN$^{(3:3)}$	81.53 ± 12.23	77.95 ± 14.82	80.64 ± 14.25
TDCNN$^{(3:3)}$	70.35 ± 13.97	67.28 ± 16.38	71.36 ± 15.36
TDCNN-RNN$^{(3:3)}$	82.87 ± 11.95	73.84 ± 15.98	82.50 ± 13.94

heated areas. The heatmaps of TDCNN-RNN$^{(1:1)}$, TDCNN$^{(1:1)}$, TDCNN-RNN$^{(2:2)}$ and TDCNN$^{(2:2)}$ are shown in Appendix C.

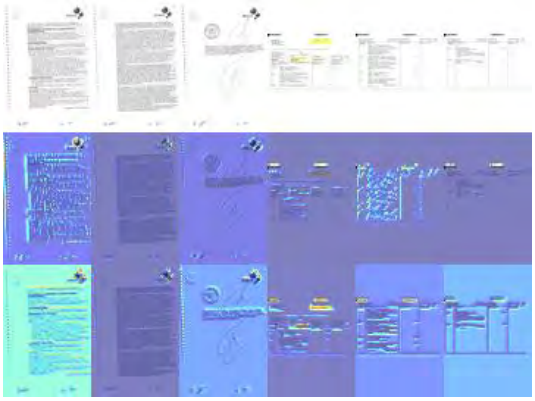
6.3 Error analysis

The error analysis looks at the different types of error the SVM and deep learning architectures make. The analysis consists of three parts, namely, the page classification error, the file splitting error and the document extraction error.

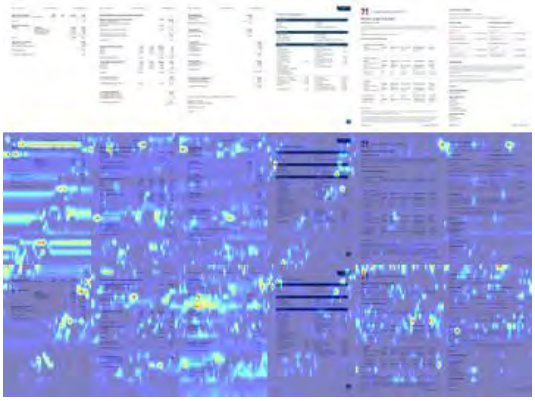
6.3.1 Page classification error

The mean percentages of correctly classified pages per page type for the SVM and TDCNN-RNN$^{(3:3)}$ model are shown in Figure **??**. As one can see, it seems that the SVM model outperforms the TDCNN-RNN$^{(3:3)}$ model slightly for some page types. It is noticeable that the TDCNN-RNN$^{(3:3)}$ model is better than the SVM model when it comes to the 'Additional information' page type. Also, the page types for which the models almost have a perfect score are also the ones which are highly represented in the data (see Figure **??**)

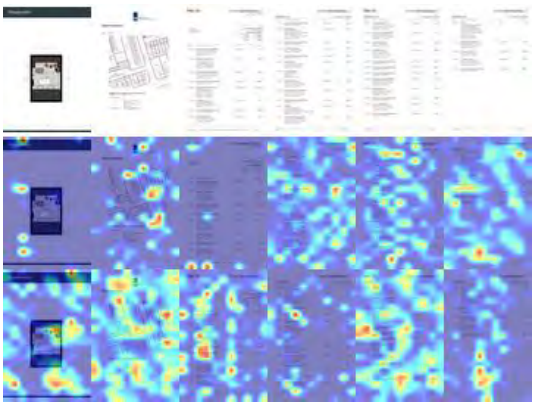
The TDCNN-RNN$^{(3:3)}$ uses the text as input. The quality of the text is therefore important for the quality of the prediction. From the error analysis, the mean number of correctly predicted page types per file (averaged over all evaluation sets) using OCR text



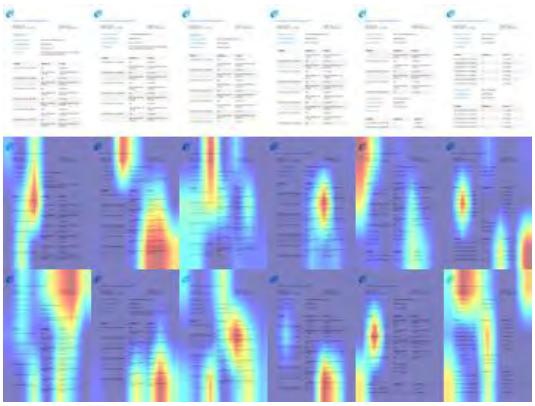
(a) Deed of Purchase - Split (conv1)



(b) Income Tax Statement - Split (conv2)



(c) Bank account Statement - No split (conv3)



(d) UWV insurance Statement - No split (conv4)

Figure 6.1: The Grad-CAM heatmap results of the four TDCNN blocks on different document type and split/no split combinations. The first row shows the input images, the second row are the heatmaps maps of the vision part of the TDCNN-RNN$^{(3:3)}$ model and the last are the heatmaps of the TDCNN$^{(3:3)}$ model.

is 85.64% (± 0.49%). The mean number of correctly predicted page types per file using the PDF reader is 91.06% (± 0.32%).

6.3.2 File splitting error

To see what type of splitting errors the SVM and TDCNN-RNN$^{(3:3)}$ models make, the confusion matrices are shown in Table 6.4. The results are computed by averaging the confusion matrices of each set (in total 100), producing the means and standard deviations. As one can see, the SVM model has a relatively large confusion between the golden 'No split' and predicted 'Split'. The SVM rarely makes a mistake between the golden 'Split' and predicted 'No split'. The TDCNN-RNN$^{(3:3)}$ seems to be better in predicting a 'No split' correctly than the SVM model. Also, the model makes fewer mistakes between the 'No split' and predicted 'Split'. It does make more mistakes between the golden 'Split' and predicted 'No split' than the SVM model.

Table 6.4: The split and no split error matrix of the SVM and TDCNN-RNN$^{(3:3)}$ models on the 100 randomly merged multi-document based on the test data set.

		Predicted (SVM)		Predicted (TDCNN-RNN$^{(3:3)}$)	
		No split	Split	No split	Split
Golden	No split	2660.10 ± 76.59	329.10 ± 5.53	2883.49 ± 75.72	105.82 ± 6.51
	Split	4.40 ± 2.25	806.89 ± 5.53	19.76 ± 3.75	791.53 ± 6.30

6.3.3 Document extraction error

As one can see in Figure **??**, the average (and standard deviation) number of times a document type got fully extracted from a multi-document file by SVM and the RNN$^{(1:1)}$ model. The SVM outperforms the RNN$^{(1:1)}$ model in most cases, in particular the 'REST' and 'Deed of transfer' document types. The RNN$^{(1:1)}$ model does extract the 'Registry map' a lot better than the SVM. Also, the standard deviation of the RNN$^{(1:1)}$ model is sometimes a bit larger than that of the SVM. This means that the performance of the RNN$^{(1:1)}$ model depends more on the multi-document files that are in a set, relatively to the SVM model.

7

Discussion & conclusion

This chapter contains a discussion about certain aspects of the research and a conclusion where the research question, and its sub-questions, are answered.

7.1 Discussion

The discussion is divided into three parts, namely, the methods, results and business applications.

7.1.1 Methods

Firstly, the data methods. To get the textual data from scanned documents OCR tooling is used which consists of a model combination that is chosen based on the benchmark results. However, the benchmark data sets are different from the data set provided by the company, in terms of content, language, form types and (lack of) colour. So the decision about which model to use, based on the results of these data sets, should therefore be taken with a grain of salt. Next, the data is generated by (pseudo)randomly merge documents together and partitioning those into chunks. The procedure that is used to generate these multi-document files has influence on the learning of the model. The procedure itself has some constraints, e.g. two consecutive documents of the same document type are not allowed. Although this procedure seems to let the model learn well from this data, other procedures might let the model learn and generalise better. Next, the modelling methods. The models described in this thesis are combinations of fixed architectures, such as the feature extractors (e.g. number of layers, activation functions) and the two classifying networks. It is very much possible that other types of feature extractors or classification networks can be used to solve the problem. Although the RNN feature

extractors performed well, a less complex feature extractor could also work in this case, e.g. a time distributed dense network/multi layer perceptron (MLP). Also, as shown in Figure 3.4 and 3.5, the textual information flow for the page classification network is not solely based on page k itself. There is a flow of information from page k-1. This can help the model since page k-1 and page k can be of the same page type or confuse it since it might wrongly use the information of page k-1. Furthermore, directly optimising a model to make split decisions does seem to work better than indirectly optimising a model, as with the SVM. Although the SVM and text works better in general than the other models, it is still useful to investigate how layout features can be incorporated into a model. This is because sometimes the OCR fails to extract words properly, returning no text, which leads to ambiguous decision-making by the model.

7.1.2 Results

The metrics which are used in this research are focused on three aspects, namely, correctly classifying document pages, correctly splitting in a document and a combination of these two. The assumption with these metrics is that each document/page/split type is equally important, which might not be the case in more real world situation. Furthermore, the error analysis shows that the models tend to over-split, in contrast to the imbalance. A reason for this could be that the (textual) content of a page can be significantly different, while belonging to the same document.

7.1.3 Business applications

7.2 Research questions

Firstly, the sub-questions are answered,

- *How can this problem be formulated as a general solvable problem?*
  Based on Chapter 3, the problem is first described as a multiclass-multioutput classification problem where the whole document is given as input. The output would be multiple decisions about whether a certain document type ends after each page or continues. The model needed for such decision-making would be fairly large, since all the information is given to the model at once. The second mathematical formulation of the problem assumes that a model can make sufficient decisions using only a part of the whole document. The results show that increasing context does not increase the performance of the model, which suggest that the assumption is reasonable. This

formulation transforms the problem into two, more mainstream, problems, namely, a binary classification problem to determine if a split is needed or not and a multiclass classification problem to determine the page type of a page. By sliding over the whole document, each page gets classified and after each page a decision if a split is needed or not is made.

- *What Optical Character Recognition methods can be used to gather text from non-readable files?*
  To get the textual data from each page one can simply use a PDF reading tool to extract it. However, old and scanned documents are seen as just images by such a PDF reading tool. To get the textual data from these type of files, one uses OCR to convert these pages into text. In Chapter 4, several off-the-shelf OCR models are tested on benchmark data sets. From the results shown in Table 4.1, the GPU accelerated deep learning models perform best. Furthermore, selecting a good text detection model seems to have a larger influence on the performance than choosing a good text recognition model.

- *What effect does the synthetic data have on the model performance?*
  As explained, each metric is the average over all the synthetically generated multi-document file sets. The standard deviation gives insight into the variety of the model performance on these sets. It is noticeable that these standard deviations can be quite large, which can mean two things, the variety of the selected documents is quite large and/or some combinations of documents are relatively hard/easy to distinguish. Also, from Table 6.4 it is clear that the splitting performance of both models depends a bit on the content of an evaluation set. So, the effect of the synthetically generated evaluation sets is that the models (SVM and deep learning models) can have a different performance, in terms of the metrics used in this thesis.

- *What type of Deep Learning architectures can be used for this problem?*
  To solve this problem, several types of architectures are described and tested, as explained in Chapter 3 and 6. The general concept is to use context and automatically extract features from the graphical representations (CNN based) and textual representations (TF-IDF and RNN based), which are then used to classify a page and make a split decision. The main assumption with this modelling method is that close context (i.e. neighbouring pages) are enough to make a decision. From the results in Table 6.1, 6.2 and 6.3, it seems that adding more context does not increase

the score by much and does in some cases cause more confusion, resulting in a lower score.

- *Which parts of the input data does the model base its decisions on?*
  From the heatmaps shown in Figure 6.1, it seems that the first convolutional block does pick up on certain logos, stamps and other logical visually rich areas. The other convolutional blocks show a rather more abstract type of vision. It is interesting to see that, in the given examples, all pages contribute to the models decision-making, instead of a few pages contributing a lot. It is hard to determine if the models always look at logical visually rich areas, and use those information bits to determine if pages are from the same (sub)document.

Answering the main research question,

To what extent can Convolutional and Recurrent Neural Networks be used to partition Dutch mortgage files based on their graphical and textual content into separate logical mortgage documents?

From the experiments in Chapter 6, it is clear that the SVM baseline, using only textual data, is a good contender to solve this problem. Also, from testing several combinations of CNN and/or RNN architectures it seems that adding extra context does not necessarily increase performance. It is clear from Table 6.1, 6.2 and 6.3 that solely using the graphical data is not sufficient to solve this problem. When looking at the sub-document extraction test scores, $o_{\text{doc}}^{\textit{sub-document}}$, it is clear that it is in general hard to fully correct extract all the sub-documents in a given file. The best sub-document extraction score is on average (over all evaluation sets) 85.99%, meaning that on average 85.99% of the sub-documents in a file are correctly fully extracted. Thus, it is certainly possible to split files and classify the sub-documents using the methods described in this thesis. Although, there is room for improvement.

7.3 Future research

References

[1] Abdelrahman Abdelhamed, Stephen Lin, and Michael S Brown. A high-quality denoising dataset for smartphone cameras. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1692–1700, 2018. 6

[2] Muhammad Zeshan Afzal, Andreas Kölsch, Sheraz Ahmed, and Marcus Liwicki. Cutting the error by half: Investigation of very deep CNN and advanced training strategies for document image classification. *CoRR*, abs/1704.03557, 2017. 6

[3] Onur Agin, Cagdas Ulas, Mehmet Ahat, and Can Bekar. An approach to the segmentation of multi-page document flow using binary classification. In *Sixth International Conference on Graphic and Image Processing (ICGIP 2014)*, volume 9443, page 944311. International Society for Optics and Photonics, 2015. 7, 16

[4] Sanghyeon An, Min Jun Lee, Sanglee Park, Heerin Yang, and Jungmin So. An ensemble of simple convolutional neural network models for MNIST digit recognition. *CoRR*, abs/2008.10400, 2020. 6

[5] Srikar Appalaraju, Bhavan Jasani, Bhargava Urala Kota, Yusheng Xie, and R. Manmatha. Docformer: End-to-end transformer for document understanding. *CoRR*, abs/2106.11539, 2021. 6

[6] Enrico Appiani, Francesca Cesarini, Anna Maria Colla, Michelangelo Diligenti, Marco Gori, Simone Marinai, and Giovanni Soda. Automatic document classification and indexing in high-volume applications. *International Journal on Document Analysis and Recognition*, 4(2):69–83, 2001. 8

[7] Stefano Baldi, Simone Marinai, and Giovanni Soda. Using tree-grammars for training set expansion in page classi. cation. In *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, pages 829–833. Citeseer, 2003. 8

[8] Christopher Bowles, Liang Chen, Ricardo Guerrero, Paul Bentley, Roger N. Gunn, Alexander Hammers, David Alexander Dickie, Maria del C. Valdés Hernández, Joanna M. Wardlaw, and Daniel Rueckert. GAN augmentation: Augmenting training data using generative adversarial networks. *CoRR*, abs/1810.10863, 2018. 29

[9] Quang Anh Bui, David Mollard, and Salvatore Tabbone. Automatic synthetic document image generation using generative adversarial networks: application in mobile-captured document analysis. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 393–400. IEEE, 2019. 29

[10] Nawei Chen and Dorothea Blostein. 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, 2007. 8

[11] Xiaoxue Chen, Lianwen Jin, Yuanzhi Zhu, Canjie Luo, and Tianwei Wang. Text recognition in the wild: A survey, 2020. 6

[12] KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014. 5

[13] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014. 5

[14] Kevyn Collins-Thompson and Radoslav Nickolov. A clustering-based algorithm for automatic document separation. In *SIGIR 2002 Workshop on Information Retrieval and OCR: From Converting Content to Grasping, Meaning, Tampere, Finland*, 2002. 7

[15] Hani Daher, Mohamed-Rafik Bouguelia, Abdel Belaid, and Vincent Poulain D'Andecy. Multipage administrative document stream segmentation. In *2014 22nd International Conference on Pattern Recognition*, pages 966–971, 2014. 7

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

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 6

[18] Yuning Du, Chenxia Li, Ruoyu Guo, Xiaoting Yin, Weiwei Liu, Jun Zhou, Yifan Bai, Zilin Yu, Yehua Yang, Qingqing Dang, and Haoshuang Wang. PP-OCR: A practical ultra lightweight OCR system. *CoRR*, abs/2009.09941, 2020. 6, 21

[19] D Eastlake 3rd and Paul Jones. Us secure hash algorithm 1 (sha1). Technical report, 2001. 28

[20] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. 4

[21] Xin Feng, Youni Jiang, Xuejiao Yang, Ming Du, and Xin Li. Computer vision algorithms and hardware implementations: A survey. *Integration*, 69:309–320, 2019. 5

[22] Jacob Gildenblat and contributors. Pytorch library for cam methods. `https://github.com/jacobgil/pytorch-grad-cam`, 2021. 20

[23] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`. 3

[24] Albert Gordo, Marçal Rusiñol, Dimosthenis Karatzas, and Andrew D. Bagdanov. Document classification and page stream segmentation for digital mailroom applications. *International Conference on Document Analysis and Recognition (ICDAR)*, pages 621–625, 2013. 7

[25] Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013. 5

[26] Erkam Guresen and Gulgun Kayakutlu. Procedia computer science definition of artificial neural networks with comparison to other networks. *Procedia Computer Science*, 00:0–000, 2010. 3

[27] Adam W Harley, Alex Ufkes, and Konstantinos G Derpanis. Evaluation of deep convolutional nets for document image classification and retrieval. In *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*, pages 991–995. IEEE, 2015. 8

[28] Adam W Harley, Alex Ufkes, and Konstantinos G Derpanis. Evaluation of deep convolutional nets for document image classification and retrieval. In *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*, pages 991–995. IEEE, 2015. 22

[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 4

[30] Pierre Héroux, Sébastien Diana, Arnaud Ribert, and Eric Trupin. Classification method study for automatic form class identification. In *Proceedings. Fourteenth International Conference on Pattern Recognition (Cat. No. 98EX170)*, volume 1, pages 926–928. IEEE, 1998. 8

[31] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 5

[32] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019. 4, 22

[33] Jing Huang, Guan Pang, Rama Kovvuri, Mandy Toh, Kevin J. Liang, Praveen Krishnan, Xi Yin, and Tal Hassner. A multiplexed network for end-to-end, multilingual OCR. *CoRR*, abs/2103.15992, 2021. 6, 21

[34] Yupan Huang, Tengchao Lv, Lei Cui, Yutong Lu, and Furu Wei. Layoutlmv3: Pre-training for document ai with unified text and image masking. *arXiv preprint arXiv:2204.08387*, 2022. 6, 22

[35] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015. 4

[36] Guillaume Jaume, Hazim Kemal Ekenel, and Jean-Philippe Thiran. Funsd: A dataset for form understanding in noisy scanned documents. *ICDARW*, 2:1–6, 2019. 22

[37] Le Kang, Jayant Kumar, Peng Ye, Yi Li, and David Doermann. Convolutional neural networks for document image classification. In *2014 22nd International Conference on Pattern Recognition*, pages 3168–3172, 2014. 8

[38] Jasmin Kurtanović. Deep learning - ocr text detection and text recognition, 2021. 6, 21

[39] Minghui Liao, Zhaoyi Wan, Cong Yao, Kai Chen, and Xiang Bai. Real-time scene text detection with differentiable binarization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(07):11474–11481, Apr. 2020. 22

[40] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. *ECCV - Computer Vision*, pages 740–755, 2014. 6

[41] Zachary Chase Lipton. A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019, 2015. 4

[42] Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, Furu Wei, and Baining Guo. Swin transformer V2: scaling up capacity and resolution. *CoRR*, abs/2111.09883, 2021. 6

[43] Shangbang Long, Xin He, and Cong Yao. Scene text detection and recognition: The deep learning era. *International Journal of Computer Vision*, 129(1):161–184, 2021. 6, 21

[44] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2017. 14

[45] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004. 7, 16

[46] Simone Marinai, Marco Gori, and Giovanni Soda. Artificial neural networks for document analysis and recognition. *IEEE Transactions on pattern analysis and machine intelligence*, 27(1):23–35, 2005. 6

[47] Mindee. doctr: Document text recognition. `https://github.com/mindee/doctr`, 2021. 21, 22, 23

[48] Rafael Padilla, Sergio L. Netto, and Eduardo A. B. da Silva. A survey on performance metrics for object-detection algorithms. In *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pages 237–242, 2020. 24

[49] Seunghyun Park, Seung Shin, Bado Lee, Junyeop Lee, Jaeheung Surh, Minjoon Seo, and Hwalsuk Lee. {CORD}: A consolidated receipt dataset for post-{ocr} parsing. In *Workshop on Document Intelligence at NeurIPS 2019*, 2019. 22

[50] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. 10

[51] Christian Reisswig, Anoop R. Katti, Marco Spinaci, and Johannes Höhne. Chargrid-ocr: End-to-end trainable optical character recognition through semantic segmentation and object detection. *CoRR*, abs/1909.04469, 2019. 6

[52] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-CAM: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128(2):336–359, oct 2019. 2, 20

[53] Baoguang Shi, Xiang Bai, and Cong Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *IEEE transactions on pattern analysis and machine intelligence*, 39(11):2298–2304, 2016. 22

[54] Christian Shin, David Doermann, and Azriel Rosenfeld. Classification of document pages using structure-based features. *IJDAR*, 3:232–247, 05 2001. 8

[55] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019. 29

[56] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 4, 22

[57] Patrick Van Der Smagt, Ben Krose, and Ben Krr. *An introduction to Neural Networks*. University of Amsterdam, 2011. 3

[58] Ray Smith. `https://github.com/tesseract-ocr/tesseract`, 2021. 22

[59] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *Computational Intelligence and Neuroscience*, 2018:1–13, 02 2018. 5

[60] Toyohide Watanabe, Qin Luo, and Noboru Sugie. Layout recognition of multi-kinds of table-form documents. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 17, 1995. 8

[61] Gregor Wiedemann and Gerhard Heyer. Page stream segmentation with convolutional neural nets combining textual and visual features. *CoRR*, abs/1710.03006, 2017. 15, 16

[62] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431, 2016. 17

[63] Yang Xu, Yiheng Xu, Tengchao Lv, Lei Cui, Furu Wei, Guoxin Wang, Yijuan Lu, Dinei A. F. Florêncio, Cha Zhang, Wanxiang Che, Min Zhang, and Lidong Zhou. Layoutlmv2: Multi-modal pre-training for visually-rich document understanding. *CoRR*, abs/2012.14740, 2020. 6, 7, 22

[64] Yiheng Xu, Minghao Li, Lei Cui, Shaohan Huang, Furu Wei, and Ming Zhou. Layoutlm: Pre-training of text and layout for document image understanding. *CoRR*, abs/1912.13318, 2019. 6, 7, 22

[65] Yiheng Xu, Tengchao Lv, Lei Cui, Guoxin Wang, Yijuan Lu, Dinei Florêncio, Cha Zhang, and Furu Wei. Layoutxlm: Multimodal pre-training for multilingual visually-rich document understanding. *CoRR*, abs/2104.08836, 2021. 6, 7, 22

[66] Syed Waqas Zamir, Aditya Arora, Salman H. Khan, Munawar Hayat, Fahad Shahbaz Khan, and Ming-Hsuan Yang. Restormer: Efficient transformer for high-resolution image restoration. *CoRR*, abs/2111.09881, 2021. 6

[67] Zhengli Zhao, Zizhao Zhang, Ting Chen, Sameer Singh, and Han Zhang. Image augmentations for gan training. *arXiv preprint arXiv:2006.02595*, 2020. 29

Appendices

Appendix A

Loss curves

The training and validation loss curves shown in Figure A.1, A.2 and A.3 are based on five runs. Note that each validation point is made by evaluating the model with the whole validation set after each epoch (apart from epoch '0'). The losses shown for the training data are made during the training phase and are based on a batch of data and not the whole training data set (in contrast to the computation of the validation losses). As one can see in Figure A.1, the classification losses for the RNN$^{(1:1)}$ start to diverge after epoch 5. The validation split loss is stable after the second epoch and the training split loss decreases a bit but also stabilises. The classification losses for the TDCNN$^{(1:1)}$ seem to diverge after the third epoch. The validation split loss does not decrease after the first epoch. Since the training split loss does decrease, a small diverge can be seen between the validation and training split loss. The TDCNN-RNN$^{(1:1)}$ seems to minimise the loss the fastest. The validation classification loss seems to be at its lowest at the third epoch, the classification losses start to diverge (more) after this epoch. Figure A.2 shows the classification and split losses for the RNN$^{(2:2)}$, TDCNN$^{(2:2)}$ and TDCNN-RNN$^{(2:2)}$. The training and validation classification loss of the RNN$^{(2:2)}$ diverge after the fifth epoch. The split loss difference of the training and validation increases after the first epoch. The TDCNN$^{(2:2)}$ seems to get overfit after the second epoch, for both the splitting and classification. The splitting loss of the TDCNN-RNN$^{(2:2)}$ diverges after the first epoch and the classification loss diverges after the second epoch. At last, Figure A.3 shows the classification and split losses for the RNN$^{(3:3)}$, TDCNN$^{(3:3)}$ and TDCNN-RNN$^{(3:3)}$. The same observations can be made as with the RNN$^{(2:2)}$, TDCNN$^{(2:2)}$ and TDCNN-RNN$^{(2:2)}$.

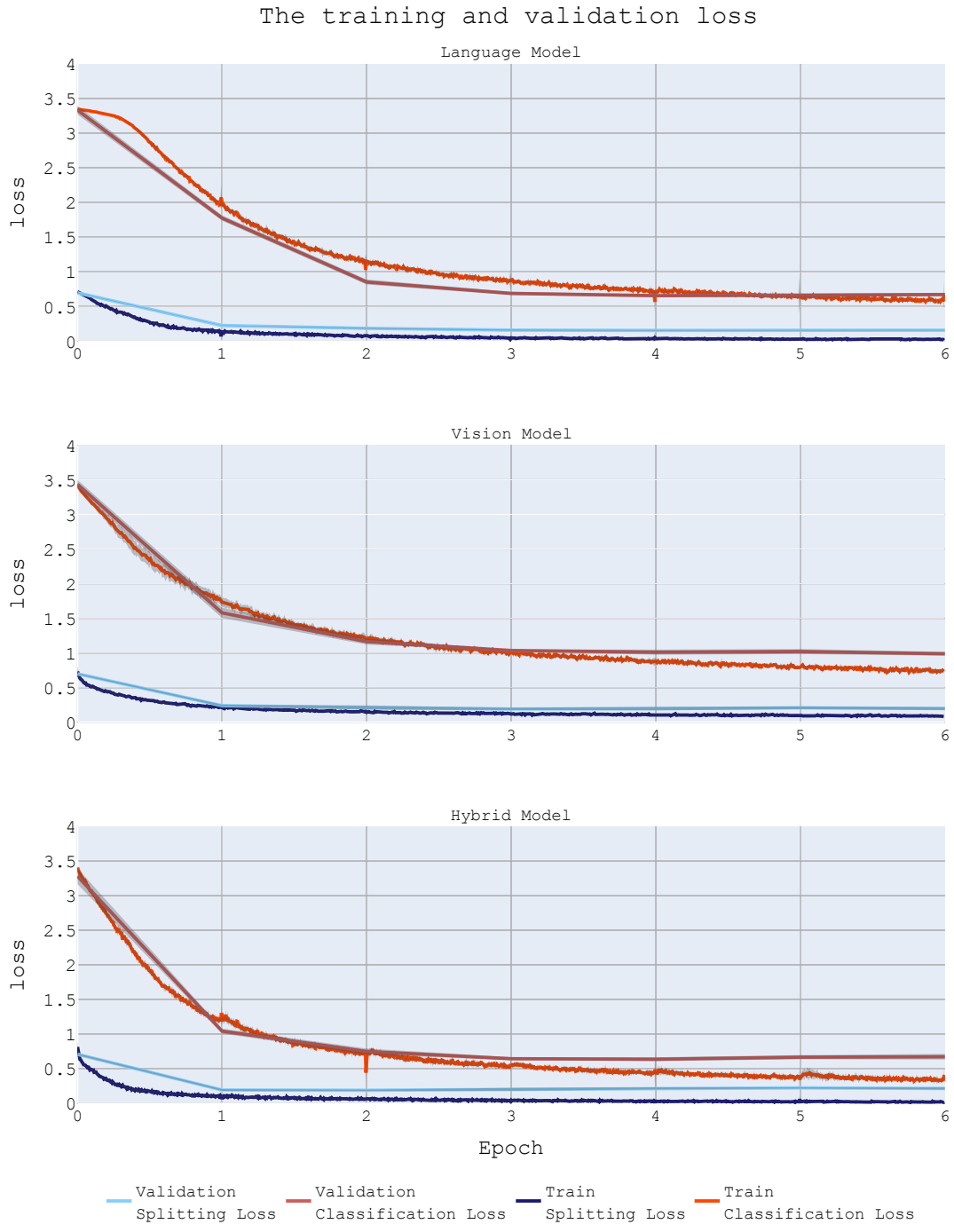


Figure A.1: The means and standard deviations of the splitting (blue) and classification (red) loss curves of the Language, Vision and Hybrid models. Note that the classification and splitting loss curves shown in the figures are multiplied by $\beta_1(=1)$ and $\beta_2(=3)$, respectively. All models are trained five times from scratch (random initialisation), using *one* front and *one* back context page as input. A learning rate of 1e-3 and a weight decay rate of 1e-1 are used together with batchsizes 1,500, 1,000 and 800 for the Language ($\text{RNN}^{(1:1)}$), Vision ($\text{TDCNN}^{(1:1)}$) and Hybrid ($\text{TDCNN-RNN}^{(1:1)}$) models, respectively.

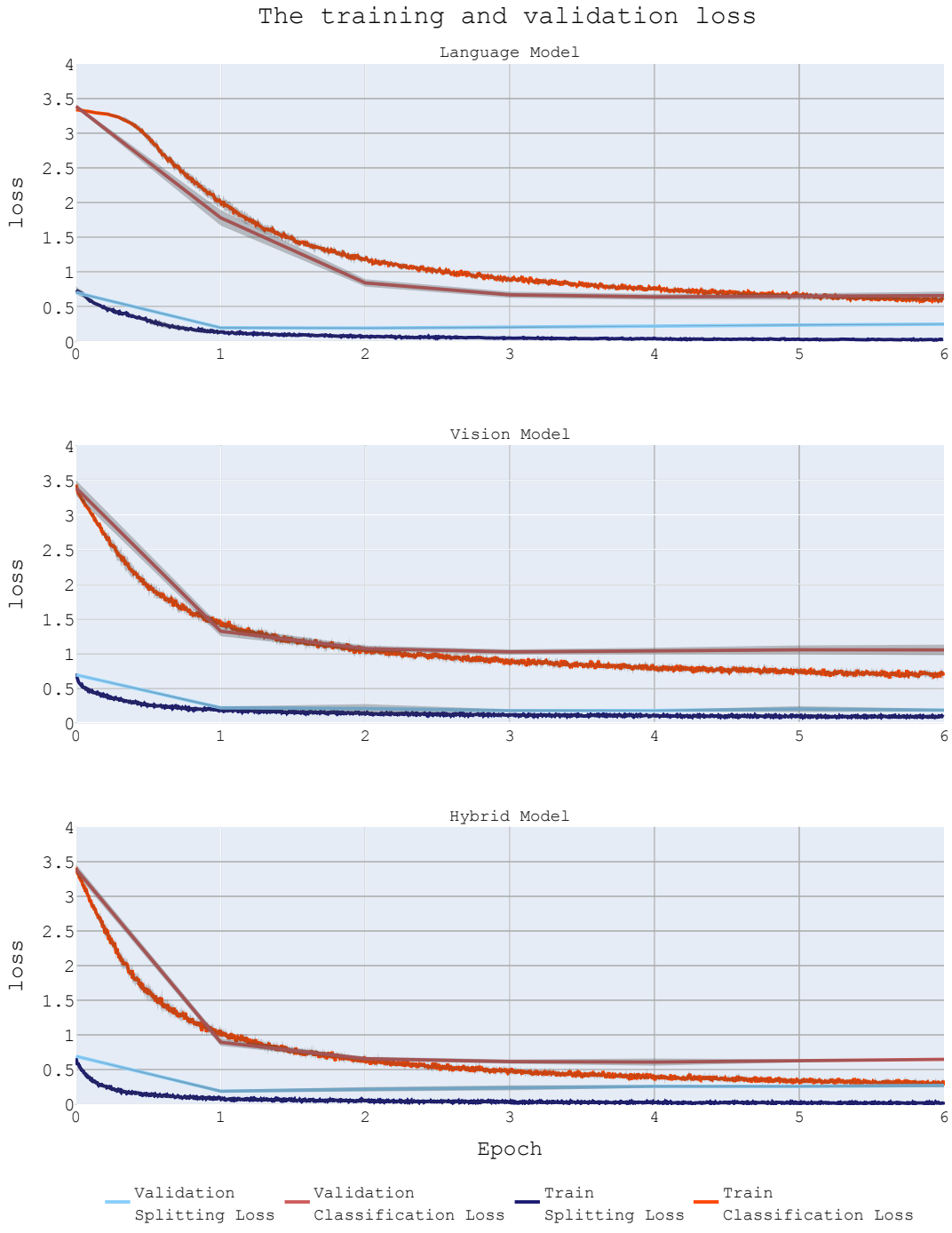


Figure A.2: The means and standard deviations of the splitting (blue) and classification (red) loss curves of the Language, Vision and Hybrid models. Note that the splitting and classification loss curves shown in the figures are multiplied by $\beta_1(=1)$ and $\beta_2(=3)$, respectively. All models are trained five times from scratch (random initialization), using *two* front and *two* back context pages as input. A learning rate of 1e-3 and a weight decay rate of 1e-1 are used together with batchsizes 1,200, 700 and 500 for the Language ($\text{RNN}^{(2:2)}$), Vision ($\text{TDCNN}^{(2:2)}$) and Hybrid ($\text{TDCNN-RNN}^{(2:2)}$) models, respectively.

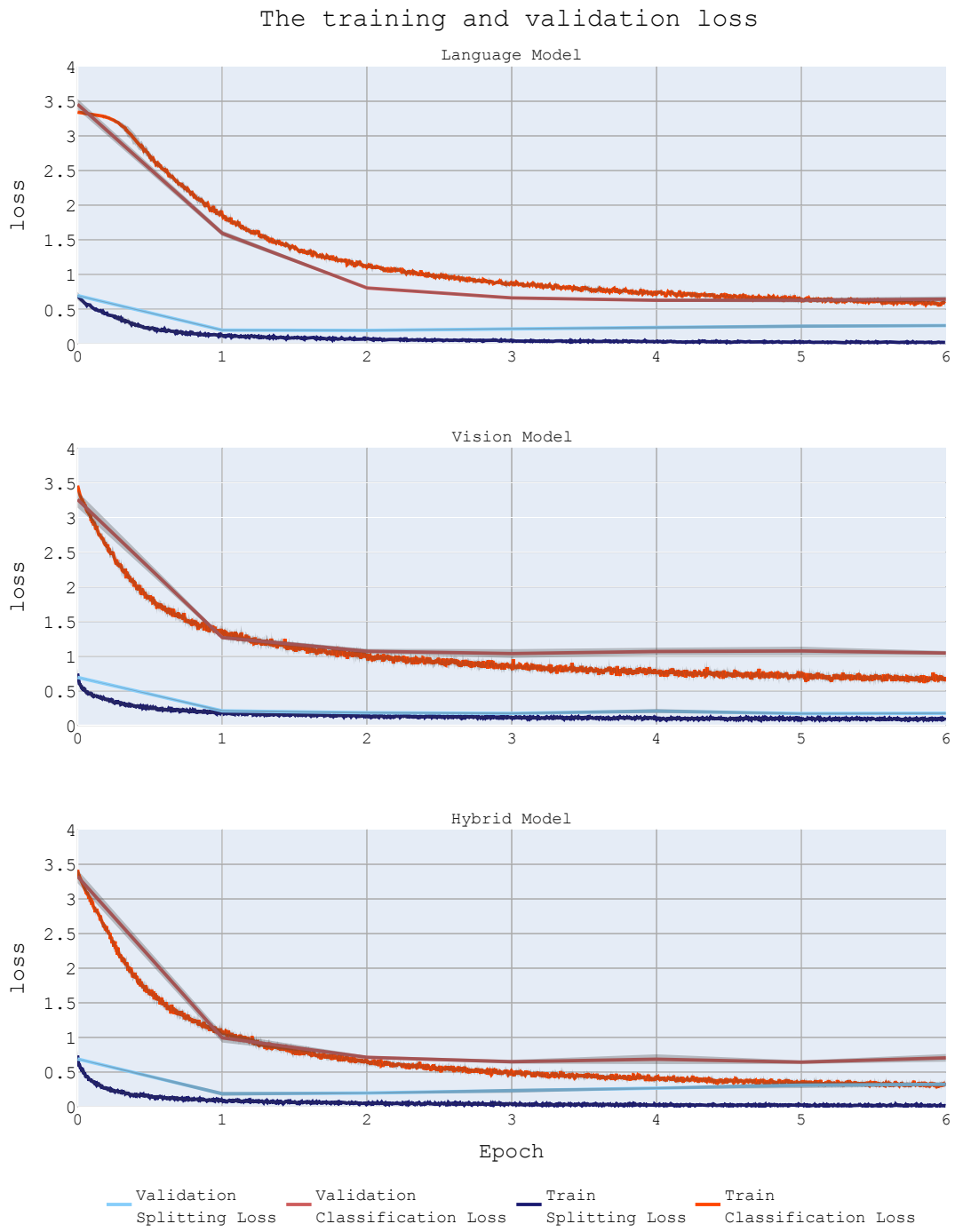


Figure A.3: The means and standard deviations of the splitting (blue) and classification (red) loss curves of the Language, Vision and Hybrid models. Note that the classification and splitting loss curves shown in the figures are multiplied by $\beta_1(=1)$ and $\beta_2(=3)$, respectively. All models are trained five times from scratch (random initialization), using *three* front and *three* back context pages as input. A learning rate of 1e-3 and a weight decay rate of 1e-1 are used together with batchsizes 1,000, 800 and 600 for the Language ($\text{RNN}^{(3:3)}$), Vision ($\text{TDCNN}^{(3:3)}$) and Hybrid ($\text{TDCNN-RNN}^{(3:3)}$) models, respectively.

Appendix B

Recall and Precision

The recall and precision of the $o_{doc}^{unnamed}$ and o_{doc}^{named} evaluation metrics are shown in Table B.1, B.2, B.3 and B.4. As one can see in Table B.1, the $o_{doc}^{unnamed}$ recall scores for the SVM model indicate overfitting. The TDCNN-RNN$^{(3:3)}$ model has the best score on the test data set. Table B.2 shows that the TDCNN-RNN$^{(3:3)}$ model has the highest precision on the test data set but the other models are only slightly worse. The SVM model has the highest recall and precision score for the o_{doc}^{named} evaluation metric, as can be seen in Table B.3 and B.4.

Table B.1: The $o_{doc}^{unnamed}$ macro recall scores of the models on the training, validation and test data in terms of means and standard deviations (in %). The context (i.e. pages before and after the split decision), if applicable, is denoted as $(m_1 : m_2)$. The average and standard deviations are computed based on the means of the evaluation set (100 in total).

Model / **Data set**	**Train**	**Validation**	**Test**
SVM	**97.89** ± 4.27	91.79 ± 10.49	90.46 ± 11.02
RNN$^{(1:1)}$	96.58 ± 5.27	**93.26** ± 7.86	92.61 ± 9.28
TDCNN$^{(1:1)}$	90.08 ± 8.02	87.63 ± 10.54	88.55 ± 9.78
TDCNN-RNN$^{(1:1)}$	96.20 ± 4.86	92.85 ± 7.57	93.37 ± 7.83
RNN$^{(2:2)}$	95.83 ± 4.83	92.32 ± 7.70	93.17 ± 7.58
TDCNN$^{(2:2)}$	90.53 ± 7.63	87.98 ± 10.21	91.31 ± 8.37
TDCNN-RNN$^{(2:2)}$	96.31 ± 4.49	91.63 ± 8.10	93.53 ± 6.94
RNN$^{(3:3)}$	93.91 ± 6.06	90.69 ± 9.20	92.43 ± 8.26
TDCNN$^{(3:3)}$	90.45 ± 7.12	87.67 ± 10.50	88.98 ± 9.37
TDCNN-RNN$^{(3:3)}$	95.56 ± 4.98	91.90 ± 7.67	**94.82** ± 5.99

Table B.2: The $o_{doc}^{unnamed}$ macro precision scores of the models on the training, validation and test data in terms of means and standard deviations (in %). The context (i.e. pages before and after the split decision), if applicable, is denoted as $(m_1 : m_2)$. The average and standard deviations are computed based on the means of the evaluation set (100 in total).

Model / **Data set**	**Train**	**Validation**	**Test**
SVM	**99.10** ± 2.20	96.02 ± 5.86	95.37 ± 6.15
$RNN^{(1:1)}$	98.60 ± 2.54	**96.91** ± 4.11	96.55 ± 5.02
$TDCNN^{(1:1)}$	94.77 ± 5.63	93.04 ± 7.12	93.76 ± 6.26
$TDCNN\text{-}RNN^{(1:1)}$	98.27 ± 2.85	96.04 ± 4.66	96.80 ± 4.35
$RNN^{(2:2)}$	97.92 ± 3.33	95.73 ± 4.87	96.11 ± 4.76
$TDCNN^{(2:2)}$	95.20 ± 5.09	93.56 ± 6.66	95.41 ± 5.29
$TDCNN\text{-}RNN^{(2:2)}$	98.33 ± 2.76	95.73 ± 4.80	96.41 ± 4.59
$RNN^{(3:3)}$	96.97 ± 4.24	95.04 ± 5.94	95.94 ± 5.02
$TDCNN^{(3:3)}$	94.70 ± 5.44	92.78 ± 7.12	94.18 ± 6.06
$TDCNN\text{-}RNN^{(3:3)}$	97.66 ± 3.47	95.34 ± 5.08	**97.06** ± 3.91

Table B.3: The o_{doc}^{named} macro recall scores of the models on the training, validation and test data in terms of means and standard deviations (in %). The context (i.e. pages before and after the split decision), if applicable, is denoted as $(m_1 : m_2)$. The average and standard deviations are computed based on the means of the evaluation set (100 in total).

Model / **Data set**	**Train**	**Validation**	**Test**
SVM	**96.83** ± 4.88	**90.29** ± 10.44	**89.11** ± 11.31
$RNN^{(1:1)}$	88.01 ± 9.90	85.46 ± 12.06	86.27 ± 12.50
$TDCNN^{(1:1)}$	77.94 ± 11.02	74.96 ± 13.44	77.32 ± 15.89
$TDCNN\text{-}RNN^{(1:1)}$	90.31 ± 7.99	86.18 ± 10.50	87.19 ± 11.11
$RNN^{(2:2)}$	84.23 ± 10.16	83.28 ± 12.95	88.41 ± 9.93
$TDCNN^{(2:2)}$	78.81 ± 13.08	73.71 ± 16.66	77.52 ± 14.72
$TDCNN\text{-}RNN^{(2:2)}$	91.54 ± 8.19	87.35 ± 11.52	88.93 ± 10.59
$RNN^{(3:3)}$	86.70 ± 9.17	86.74 ± 11.37	88.40 ± 10.73
$TDCNN^{(3:3)}$	76.43 ± 12.46	72.26 ± 16.62	76.21 ± 15.47
$TDCNN\text{-}RNN^{(3:3)}$	85.31 ± 10.64	79.51 ± 13.65	88.14 ± 10.38

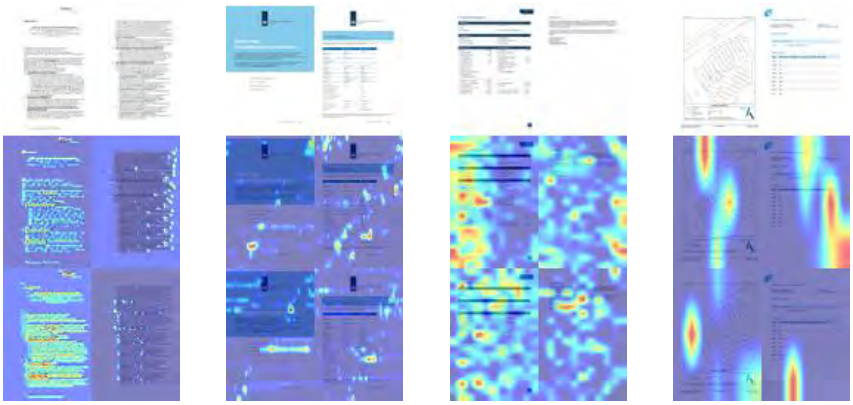
Table B.4: The o_{doc}^{named} macro precision scores of the models on the training, validation and test data in terms of means and standard deviations (in %). The context (i.e. pages before and after the split decision), if applicable, is denoted as $(m_1 : m_2)$. The average and standard deviations are computed based on the means of the evaluation set (100 in total).

Model / **Data set**	**Train**	**Validation**	**Test**
SVM	**97.95** ± 3.80	**94.77** ± 6.09	**95.19** ± 5.44
$\text{RNN}^{(1:1)}$	94.40 ± 5.30	93.35 ± 6.12	94.35 ± 5.58
$\text{TDCNN}^{(1:1)}$	85.02 ± 7.55	83.03 ± 9.24	88.19 ± 7.66
$\text{TDCNN-RNN}^{(1:1)}$	95.98 ± 4.36	92.69 ± 6.72	94.61 ± 5.57
$\text{RNN}^{(2:2)}$	91.74 ± 6.19	92.09 ± 6.65	93.49 ± 6.15
$\text{TDCNN}^{(2:2)}$	90.20 ± 5.95	86.96 ± 7.83	87.59 ± 7.73
$\text{TDCNN-RNN}^{(2:2)}$	96.39 ± 4.37	93.54 ± 6.44	94.75 ± 5.70
$\text{RNN}^{(3:3)}$	93.05 ± 6.02	93.44 ± 6.35	94.16 ± 5.89
$\text{TDCNN}^{(3:3)}$	87.92 ± 6.58	85.07 ± 8.18	88.10 ± 7.50
$\text{TDCNN-RNN}^{(3:3)}$	93.68 ± 5.11	89.74 ± 7.40	94.19 ± 5.90

Appendix C

Heatmaps

The heatmaps in Figure C.2 and C.1 show the Grad-CAM for the TDCNN-RNN$^{(1:1)}$, TDCNN$^{(1:1)}$, TDCNN-RNN$^{(2:2)}$ and TDCNN$^{(2:2)}$ models.



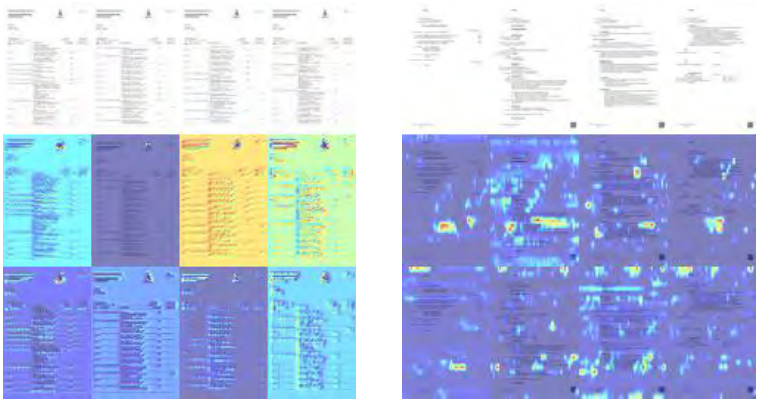
(a) Deed of transfer - No Split

(b) Energy saving report - No Split

(c) Mortgage financial report - Split

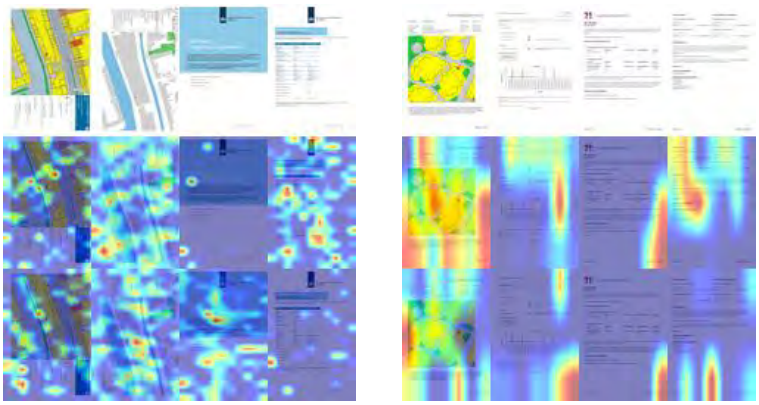
(d) Registry map - Split

Figure C.1: The Grad-CAM heatmap results of the four TDCNN blocks on different document type and split/no split combinations. The first row shows the input images, the second row are the (overlying) heatmaps maps of the vision part of the TDCNN-RNN$^{(1:1)}$ model and the last are the (overlying) heatmaps of the TDCNN$^{(1:1)}$ model.



(a) Bankaccount statements - No Split

(b) Loan contract - No Split



(c) Map - Split

(d) Test income - Split

Figure C.2: The Grad-CAM heatmap results of the four TDCNN blocks on different document type and split/no split combinations. The first row shows the input images, the second row are the (overlying) heatmaps maps of the vision part of the TDCNN-RNN$^{(2:2)}$ model and the last are the (overlying) heatmaps of the TDCNN$^{(2:2)}$ model.