Application of Deep Learning Techniques in a Commercial Document Analysis System

Dominik Bermühler

April 18, 2017
Version: 1.0

insiders technologies

Bachelor Thesis

Application of Deep Learning Techniques in a Commercial Document Analysis System

Dominik Bermühler

Supervisors apl. Prof. Dr. habil. Marcus Liwicki
Markus Ebbecke
Muhammad Zeshan Afzal

April 18, 2017

Dominik Bermühler
Application of Deep Learning Techniques in a Commercial Document Analysis System
Bachelor Thesis, April 18, 2017
Supervisors: apl. Prof. Dr. habil. Marcus Liwicki, Markus Ebbecke, and Muhammad Zeshan Afzal

TU Kaiserslautern
Computer Science
Gottlieb-Daimler-Str.
67663 Kaiserslautern

Abstract

An important feature of a Document Analysis System (DAS) is the detection of additional handwritten notes on scanned documents. Such annotations could include relevant information for the document, for example additional conditions to a contract. This thesis presents a novel approach for localising such handwritten annotations in the commercial DAS, *smart **F**or **I**nformation e**X**traction* (*smart FIX*) using Deep Convolutional Neural Networks (CNNs). To localise handwritten annotations, the approach combines different sample extraction methods and CNN classifiers. Various experiments were conducted and evaluated to find the best performing combinations of both components. It has been shown that the best obtained variant can localise handwritten annotations with a precision of up to 85 %. This outperforms the current handcrafted-feature-approach of *smart FIX* by a large margin of 13 %. During these experiments, different CNNs like *AlexNet*, *SqueezeNet*, and the novel architecture *HandDectNet* were tested on their ability to recognise handwritten annotations. Furthermore the runtime of these Deep Neural Networks were analysed in the context of *smart FIX*. As a result of this analysis, the novel architecture *HandDectNet* was developed to meet the runtime constrains of *smart FIX*. With a precision rate of 78 % and a runtime of 1.6 seconds per document, *HandDectNet* achieved better performance results than the baseline, while showing a comparable runtime.

Contents

1 Introduction

1.1 Motivation

Despite the wide spread of electronic media a large proportion of todays business communication still relies on printed documents. For a business with up to thousands or even millions of daily incoming documents, this poses a big challenge. Every incoming document has to be categorised, digitalised and entered into an information management system. While in the past this task needed a tremendous amount of workforce, it is supported by intelligent DASs today. Their task is to automatically sort a document in different categories like invoice or withdrawal, extract the relevant information and pass them to subsequent systems (for example a document management system).

Although most of the documents only contain machine printed text, sometimes they are annotated with handwritten notes. Such notes could be a remark to an important contract detail or even an additional contract clause. For a human it is easy to spot such annotations and distinguish them from machine printed text, graphics or just scan artefacts, but for a machine this task is not straight forward.

1.2 Concepts

Enabling machines to “see” and recognise objects in images is a topic of research for many years now. A common approach for object recognition tasks, like the detection of handwritten annotations, is to use classification algorithms. These classifiers learn to distinguish between objects from two or more classes, based on a training set of examples. During training, a classifier tries to correctly predict the class of the training samples and correct itself based on its prediction errors. To distinguish between samples of different classes it needs to analyse their features. Often these features are defined by a human and should capture the characteristics of the classes clearly, such that a classifier can easily distinguish samples of different classes.

While manually created features still play an important role, different approaches from the machine learning subfield Deep Learning gained a lot of attention lately.

Deep Learning algorithms are based on Artificial Neural Networks (ANNs), which are computer models inspired by the structure of biological neural networks. In the recent years, complex ANN architectures called “Deep” ANNs, led to a significant progress in classical machine learning problems, like machine translation, speech recognition, image object recognition and many more. One of their advantages is their ability to automatically extract features from given training samples. Especially in object recognition tasks, this leads to better classification results and at the same time reduces the effort necessary for manually constructing these features.

1.3 Contribution

In the scope of this work, Deep ANNs are used to localise handwritten annotations on business documents. To localise the region on a document which contains handwritten annotations, it is necessary to extract relevant areas from the document. The component responsible for this, is called extractor. It will search for small image parts in the scanned document which could contain handwritten text and extracts them from the document. After the extraction, an ANNs architecture classifies these image patches as “handwritten” or “none-handwritten”. As part of this thesis, different combinations of extraction methods and ANNs architectures were developed. Each combination (called locator) was tested and the influence of its components on each other was studied in regards of classification performance and runtime. Based on their F-measure rate, all locators were ranked. The Top-5 locators were compared to the existing, hand crafted feature approach of *smart FIX*, by using various classification and localisation performance metrics.

The main contributions of this thesis are:

- Implementation and integration of a Deep Learning based approach for detecting handwritten annotations into *smart FIX*.
- Implementation of four extractors to extract image patches from a document: *Sliding Window*, *Connected Components (*CCs*)*, *Quadtree*, *Line Segments*.
- Design of the novel CNN architecture *HandDectNet* for recognising handwritten text in images.
- Performance evaluation of the ANN architectures *AlexNet*, *AlexNet without fully connected layer (FCL)*, *SqueezeNet*, and four versions of *HandDectNet*.
- Evaluating the performance of different extraction methods and ANN architectures for finding handwritten annotations on documents. The results show that the *Line Segment* extractor in combination with *AlexNet without FCL* outperform the baseline with an precision rate of 85 % vs. 72 %.

1.4 Thesis Structure

Chapter 2 provides a brief overview over the foundations of the used methods in this thesis. It briefly introduces ANNs in general and *AlexNet*, as well as *SqueezeNet* in particular. Additionally, an overview of *smart FIX* and its functionality is given.

Chapter 3 gives the reader an excerpt from the literature of different approaches for localising handwritten annotations on documents. Additionally, the current approach for annotation detection in *smart FIX* is described.

Chapter 4 explains the structure and elements of the proposed approach. It illustrates the different extractors for extracting relevant image parts from a document, as well as the modifications made to the used ANNs. Beside of that, the novel CNN architecture *HandDectNet* is presented here.

Chapter 5 describes the data used for training, validation and testing the neural networks. It also shows the experiment setup and introduces the used metrics.

Chapter 6 evaluates the experimental results by analysing the performance of the different extractors as well as classifiers. It also gives a comparison between the five best performing extractor / classifier and the baseline. The best combinations of extraction method and ANN classifier are analysed in terms of their performance and runtime and the pros and cons of them are highlighted.

Chapter 7 summarises the results of this thesis and presents the best performing locator. Moreover the possible future work is outlined.

2 Background

This chapter gives a brief overview over the background of the methods used in this thesis. The first section introduces the topic Machine Learning as well as its popular branches. In Section 2.2, ANNs are introduced and their basic components will be explained. Section 2.3 details the CNN architectures *AlexNet* and *SqueezeNet*, used for the classification task in this thesis. In Section 2.4 an overview of the general structure and functionality of *smart FIX* is given. Especially the image preprocessing components used in this thesis are detailed, due to their importance for the approach of this thesis.

2.1 Machine Learning

Machine Learning is a branch of artificial intelligence, which studies algorithms that learn from and recognise patterns in data. It can be divided into three major branches: *Supervised Learning*, *Unsupervised Learning*, and *Reinforcement Learning*, which are shortly introduced in the following sections. In the context of this thesis, classification algorithms of the branch *Supervised Learning* are used.

Supervised Learning methods learn patterns from data through a training with labelled examples. For every training sample, the algorithm tries to predict its correct label. Based on its prediction error, it will modify its prediction function such that the error decreases.

More formally we can say that *Supervised Learning* deals with algorithms, which learn to produce an output vector $\vec{y}$ based on a training set of inputs. This training set contains different variants of input vectors $\vec{x_i}$ with their corresponding optimal target vector $\vec{t_i}$ also called label. Formally such an algorithm tries to learn the parameter vector $\vec{\theta}$ of a function $f(\vec{x}, \vec{\theta}) = \vec{y}$ such that it can map an input vector $\vec{x_i}$ to an output vector $\vec{y_i}$. This learning process is often modelled as an optimisation problem, whereas an error function $L(\vec{y_i}, \vec{t_i})$ has to be minimised. $L(\vec{y_i}, \vec{t_i})$ measures the deviation of $\vec{y_i}$ from the corresponding target output or label $\vec{t_i}$. If the output vector contains discrete values this task is called *classification*. For continues output values this is called *regression*.

Unsupervised Learning algorithms try to learn a structural representation of unlabelled training data. Clustering is one area of application, in which the training data is grouped in a given number of classes based on their similarity. This similarity is formally expressed with a function $d(\vec{x}, \vec{y}) = s$, which measures the distance between two input vectors $\vec{x}$ and $\vec{y}$. Inputs with a small distance to each other will be assigned to the same class or cluster.

Reinforcement Learning studies problems in which a software agent has to find the best action based on the current environment state and its previous actions. Thereby it will change its behaviour based on the response from the environment to its action. The quality of the agents actions are measured with a reward function, which it tries to maximise. This leads to an adaption of the agents behaviour to its environment.

2.2 Artificial Neural Networks

An Artificial Neural Network (ANN) is a simplified computer model of a biological neural network, which can be found in the brain of many organisms. Like its biological counterpart it consists of neurons or nodes, which are connected with each other through one or more directed edges. Nodes in a neural network can send signals to each other via these connections. Every connection has a weight associated with it, which determines how well a signal is broadcasted over it. A lower weight will decrease the signal, whereas a higher weight will increase it. A neural net processes a signal by passing it from one node to another via their connections. When a node receives the signals from other nodes, it aggregates their weighted signals and applies a so called activation function to the result. This function acts as a dynamic threshold, which let the signal pass if a given activation level is reached.

Although ANNs can be of any arbitrary graph structure, this thesis will only make use of ANNs without cyclic paths. These *feedforward* neural networks pass a signal from the first nodes through the network to the output nodes at the end. These kind of ANNs are often structured in different layers, in which each layer consists of many nodes. The different layers are connected via the edges between their neurons. The first and last layer are called input / output layer, while the layers in between are called hidden layer. The process of passing a signal through the network is also called *forward pass*. Such a simple ANN is shown in Figure 2.1.

A *forward pass* through an ANN with one hidden layer, like the one in Figure 2.1, can be described formally with Equation 2.1. The parameter $\vec{x}$ describes the inputs of the network, whereas $\vec{w}$ denotes all edge weights of the network grouped together

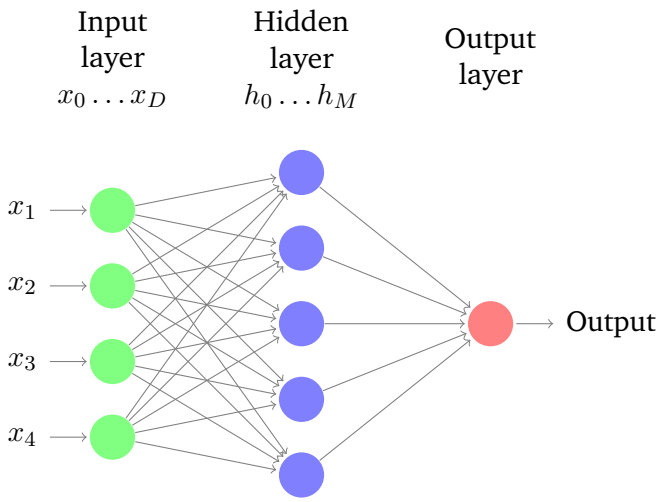


Fig. 2.1: Example of an ANN (Diagram was derived from [1])

in one vector. The variable x_i describes the i-th input neuron, while the index j indexes the neurons of the hidden layer. w_{ji} is the weight between the i-th input and the j-th hidden neuron. w_{0j} is the weight between each hidden neuron and the output neuron.

$$f(\vec{x}, \vec{w}) = \sigma\left(\sum_{j=0}^{M} w_{0j} \cdot \sigma\left(\sum_{i=0}^{D} w_{ji} \cdot x_i\right)\right) \tag{2.1}$$

The function $\sigma(a)$ acts as the threshold function. It will only let signals pass, which exceed a given threshold. Some of the more popular activation functions are the sigmoid, hyperbolic tangent or rectify function. A common activation function is the rectify function (see Equation 2.2).

$$\sigma(a) = \max(0, a) \tag{2.2}$$

To classify some input data with the help of a feedforward ANN into two or more classes, the aforementioned *forward pass* will be used. Every element of the input vector $\vec{x}$, which could be the pixel value of an image, represents the start value of one of the input neurons. This vector gets passed through the network until it reaches the output neurons. These neurons represent the classes which the ANN should distinguished. Depending on the input vector and the current weights of the edges of the network, the output neurons will have different values. Since each output neuron represents a target class, the predicted class is often determined by choosing the output neuron with the highest activation value.

In order to achieve a correct classification result, the weights of the ANN have to be chosen accordingly. This is done by training the ANN in a *Supervised Learning* manner. For each training iteration, training data annotated with the correct class label is passed through the network. Based on the classification output of the network and the correct label an error value is computed with an error function. Depending on how worse the network classified a training sample, the weights of every connection will be modified. Based on its influence on the classification error, every weight will be changed such that the error is decreased. The algorithm which performs these weight changes is called *backpropagation*.

2.2.1 Fully Connected Neural Network

Most of the time ANN are structured in layers. Every layer consists of a given number of neurons and is connected to another layer. In a Fully Connected Network (FCN) every node of one layer is connected to every other node of the next layer. An input or data vector is passed from the input layer to the first hidden layer and so on until it reaches the output layer. The output layer will often show a probability distribution over the classes in which the input vector should be classified in. This is realised through the *Softmax Layer*, which computes the probability for every output o_i with Equation 2.3.

$$f(o_i) = \frac{e^{o_i}}{\sum_{j=0}^{J} e^{o_j}} \tag{2.3}$$

2.2.2 Convolutional Neural Network

A Convolutional Neural Network (CNN) is an ANN specialised on image recognition tasks. To do so, they learn image filters during their training. Image filters are an important method in computer vision to generate features from images. Applied to an image these filters will amplify some characteristics of the image, while suppressing others.

A filter is a two dimensional weight matrix also called image kernel or window. It is applied to an input image by sliding the kernel in a given stride from the top left to the bottom right over the image, where stride refers to the number of pixels between each step. In every step all pixels which are covered by the kernel gets multiplied with the corresponding kernel weight and are subsequently summed up. The result will be the value of the pixel at the kernel position in the output image. An example for the application of a filter on an image can be seen in 2.2

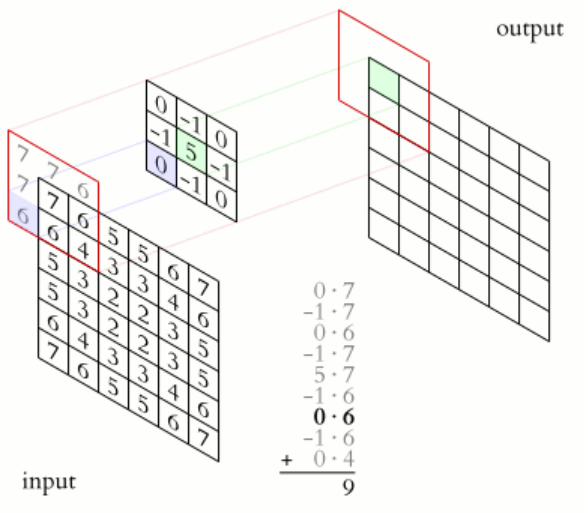


Fig. 2.2: Applying an 3×3 kernel to an image. The cyan pixel in the output image is calculated and will have the value 9. Note that the values of the border pixels in the input image are mirrored to preserve the input image size in the output image. Source: Plotke [6]

In previous approaches these filters were made by choosing the weights and their arrangement in the kernel by hand. CNNs, on the other hand, are able to learn such filters automatically from the training data. Like in a fully connected network the neurons of one convolutional layer (L_n) are connected to the neurons of the next convolutional layer (L_{n+1}). Unlike in FCNs the weight of an edge is not unique. Instead, a set of weights is shared by the edges between L_n and L_{n+1}. These weights can be seen as the weights of an image kernel, which is slided over the original input image to extract features from it. Sequentially connected convolutional layer will form a hierarchy of these filters. Through the training of the network, the first convolutional layers will mostly learn to extract simple features like edges or geometric shapes. These extracted features or filtered images are the input for following convolutional layer, which will combine them to more and more complex features. Filters learned in the last convolutional layers sometimes produce features, which can be directly linked to semantic meaning like for example faces.

Another important element of a CNN, is the pooling layer. Similar to the convolutional layer, a pooling layer is applied by sliding a window over the input image. The pooling layer will apply an aggregate function like *max* or *average* on the input pixels it covers in each step. In CNNs a convolutional layer is often followed by a max-pooling layer, to reduce the image size and to make the network translation invariant.

2.2.3 Deep Learning

To learn complex features like the ones mentioned in Section 2.2.2, the depth (or number of subsequently connected layers) of an ANN plays an important role. With a deeper ANN it is possible to learn a more complex representation of the training data, which is necessary to perform complex classification tasks like object recognition etc.

Because of the computational highly expensive matrix / vector multiplications, which are performed during the classification and training of the network, the depths of ANNs was restricted to only a few layers in the past. Through the technological advancement in hardware, especially Graphics Processing Units (GPUs), this changed in the recent years. Due to their on matrix operations specialised GPUs it is now possible to construct architectures with many layers. These deep networks made it possible to learn high level abstractions of data, like the semantic feature filters mentioned in Section 2.2.2.

2.3 Convolutional Neural Net Architectures

In the context of this thesis, deep CNNs will be used to recognise handwritten text in extracted image patches of a document. Two of these networks are *AlexNet* [4] and *SqueezeNet* [2], which will be presented in the following sections.

2.3.1 AlexNet

AlexNet [4] is a CNN which marked a breakthrough for ANNs in the field of image recognition. In 2012 it won the object recognition competition, ImageNet with remarkable results and outperformed the previous approaches by a large margin. The task of this competition is to categorise the content of an image in one of 1000 classes, with a provided training set of 1.2 million labelled images.

One of the reasons for its success is its relative deep architecture. The first five out of eight layers solely consisted of convolutions. As mentioned in Section 2.2.2 these layers construct a feature filter hierarchy, which is learned through the back propagation algorithm from the training data alone. For classification this hierarchy will output a filtered version of the input image, which is interpreted as a feature vector. This vector is classified by three fully connected layers and result in a 1000 dimensional output vector. The elements of this vector represent the classification result for every class of ImageNet.

Another reason for its good performance is the use of data augmentation. Despite the already huge training set size of 1.2 million images overfitting was still a problem during the training of *AlexNet*. Overfitting is a common problem in the training process of deep ANNs. These ANNs tend to overfit due to the fact that the neural network embodies a complex predictive function with many parameters (edge weights). With less training data such a function memorise the shown data instead of generalising from them. To prevent *AlexNet* from overfitting its training set was artificially enlarged through different label-preserving transformations. These transformations included mirroring the actual images or randomly cropping smaller patches from them. Another technique used to tackle the problem of overfitting was dropout. This technique will randomly set the output of a given number of neurons of a layer to zero in every training iteration. By zeroing out the output of a neuron it will not take part in the forward and backward pass. In this way a different network architecture is sampled in every training iteration. Dropout forces the network to learn more robust features, which are useful for different network architecture variants.

2.3.2 SqueezeNet

Iandola et al. [2] presented *SqueezeNet* as an *AlexNet* alternative, to tackle the problem of its huge number of weights. *AlexNet* has 60 million weights from which each, if saved as a float, has a size of 32 bits. This results in a complete weight size of 240 MB. Such a large memory footprint can result in problematic consequences, when multiple instances of *AlexNet* are loaded on the same machine. With *SqueezeNet*, Iandola et al. [2] tried to reduce the number of weights, while at the same time achieve the classification performance of *AlexNet*.

The number of weights for *SqueezeNet* were reduced by following three design guidelines /strategies:

1. Reduce the weights of a convolutional layer by transforming its 3×3 filter to 1×1.
2. Decrease the number of input channels to convolutional layer consisting of 3×3 filters.
3. Downsample late in the network to preserve large feature maps as long as possible.

The idea behind the first strategy arises from the fact that 3×3 is a common filter size for CNNs. It seems obvious that a reduction of weights can be achieved by

replacing these 3×3 filters, whenever possible, with 1×1.

The second strategy is derived from the following consideration: An image with n channels (in the following called feature maps) is processed by a convolutional layer by convolving every input feature map with at least one unique filter. Thus, n feature maps result at least in n filters in the convolutional layer. With the common 3×3 filter dimension this equals to a quantity of $number\ of\ input\ channels \cdot number\ of\ filters \cdot 3 \cdot 3$ weights. In *SqueezeNet* a reduction of the feature maps is achieved by inserting a *Squeeze Layer* before 3×3 layers. A *Squeeze Layer* consists of 1×1 filters only, which obviously have a smaller number of weights than 3×3 filters. Since the number of output feature maps can be controlled for each convolutional layer individually, it is possible to reduce the number of output feature maps for the *Squeeze Layer*.

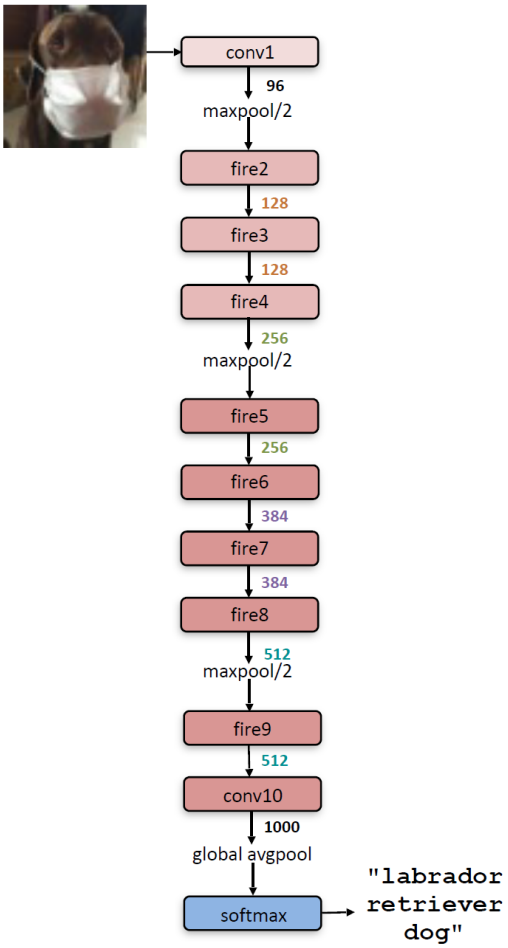


Fig. 2.3: Architecture of *SqueezeNet*. After each layer the number of output feature maps of this layer is denoted. Source: Iandola et al. [2]

This leads to a much smaller number of input feature maps for the subsequent 3×3 layer. The third and last strategy advices to downsample late in the network. This prevents the loss of valuable visual information, which enables the convolutional layers to learn better feature filters. In the end this should lead to a higher classification accuracy.

These three strategies are employed within SqueezeNet in the fire module/layer (see Figure 2.4). The first element of this module is the *Squeeze Layer*. It solely consists of 1×1 filters, which embodies the first strategy and second strategy. It reduces the number of input channels to the second part of the fire module, the *Expand Layer*. As explained earlier this is achieved by choosing a smaller number of output feature maps for the *Squeeze Layer* in comparison to the number of input feature maps. The *Expand Layer* consists of a mixture of 1×1 and 3×3 layers, where the 1×1 filter expresses strategy one.

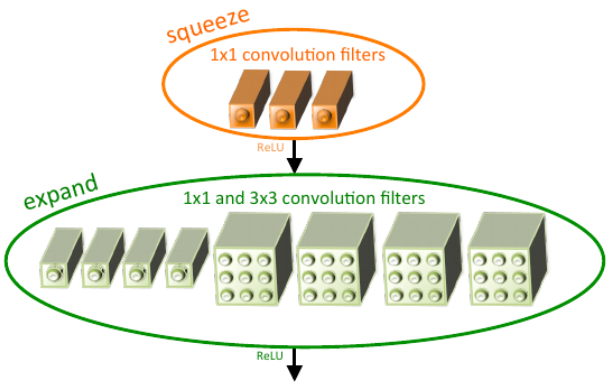


Fig. 2.4: Fire module of *SqueezeNet*. Source: Iandola et al. [2]

Most of the layers used in *SqueezeNet* are the fire modules. They are linear connected to each other (see Figure 2.3) and are the building blocks of *SqueezeNet*. The output of the *Expand Layers* often results in many feature maps, which are reduced by the *Squeeze Layer* of the subsequent fire module. This leads to a reduction of the input feature maps for the following *Expand Layer*.

2.4 smart FIX

smart FIX is a commercial DAS which classifies documents and extracts information from them e.g. invoices, medical bills, orders, etc. It can handle hundreds of thousands documents per day and is used by businesses of different industries like banking, health insurance, energy supply and others.

2.4.1 Workflow

smart FIX consists of different modular components, where each module fulfils another task in the *smart FIX* workflow. Before any classification and information extraction can be done, the user has to define the document classes which should be recognised by the system. This is done via the *DocumentManager*. Here the user defines, tests, and optimises the needed document classes. The definition of the information fields, which should be extracted for each class, is also done here.

Fig. 2.5: Workflow of *smart FIX*

After the document classes are defined, *smart FIX* is able to start the information extraction. The first step in the workflow is handled by the *Importer*. This component will import documents from arbitrary sources into the system. After this step the *Anal-*

yser will preprocess images, perform Optical Character Recognition (OCR), classify them, group them to multi-paged documents, and extract the relevant information based on the document class definition. Subsequently, it will check the extracted information against constrains that the user had defined in the DocumentManager to prevent the extraction of false information. If it is not confident enough about the correctness of the extracted information, it will pass them to the *Verifier*. Here the uncertain fields will be verified by a human. After this step, the information will be retrieved by the *Exporter*, which saves it to a database, a file or any other subsequent system.

2.4.2 Preprocessing

The preprocessing is part of the *Analyser*. It has the task to clean the document image from artefacts and to prepare it for the subsequent information extraction. The preprocessing consists of the following steps:

At first, the *Analyser* binaries the input image. This means it will convert a colour image into a black and white image, to ease the following analysis. In the second step, it recognises and corrects skewed images, as well as images which are not correctly oriented. Often documents like forms contain help lines, which specify the location on the document where handwritten text has to be placed. These horizontal and vertical lines, as well as other irrelevant elements like scan artefacts, are removed in the next step. Similar to the previous step, it is also possible to remove CCs[1] bigger than a specified size.

After the image was cleaned from any disturbing elements, the remaining black pixels get segmented into lines, words, and characters. These are the input of the OCR, which finally reads the text on the image.

[1] A cluster of neighbouring black pixels. A more detailed explanation will be provided in Section 4.2.2.

3 Related Work

In this chapter three feature based approaches for handwriting detection will be presented. These approaches are using different features and classification methods to distinguish between handwritten and machine printed text. Section 3.1 and 3.2 show two approaches from the current literature. The last Section presents the existing handwriting localisation approach of *smart FIX*.

3.1 Upper-Lower Line Profile

Kavallieratou and Stamatatos [3] used the uniformity respectively the difference of the character height of machine printed and handwritten text lines to distinguish them. Based on this characteristic they defined three features which were used to build a classifier to discriminate handwritten and machine printed text lines.

Fig. 3.1: b) and e) show the upper-lower line profile of the text lines a) and d). c) and f) are the vertical histograms of their corresponding lines. Source: Kavallieratou and Stamatatos [3]

The features used for the classification are constructed from an upper-lower profile of each text line. An upper-lower profile contains the highest and lowest black pixel of each pixel column. If plotted, these pixels result in an outline of the text (see Figure 3.1). In the next step a horizontal histogram of the black pixels in dependency of the line height is calculated from this profile (see b) and f) in Figure 3.1). With the help of this histogram, the text line can be divided into three regions: upper, lower, and middle region. The upper region is defined as the area above the estimated upper character line. This line is represented by the most significant spike of the histogram above the middle pixel line. Similar to the upper region, the lower region is defined as the area under the estimated bottom character line. This line is represented by most significant spike of the histogram below the middle pixel line of the text line. The region between the upper and bottom character line is the middle region. From these regions the following features are constructed: The first feature is the ratio between the height of the upper region and the height of the middle region. The second feature is defined as the ratio between the height of the lower region and the middle region. The last feature is the ratio between the area and the maximum

value of the vertical pixel histogram. With these three features a linear discriminant analysis was conducted to construct a linear classifier.

3.2 Eigencharacters

Pinson and Barrett [5] presented another approach, which applies the method of "Eigenfaces" [7] to discriminate handwritten from machine printed CCs. It uses a Principal Component Analysis (PCA) to project CCs of single characters to a vector space called "character space". In this vector space, characters sharing similar statistical characteristics are close to each other. For classification Pinson and Barrett placed 5957 representative machine printed characters in the character space, together with a set of handwritten and additional machine printed CCs. To determine the class of a new data sample a local distance threshold for each representative character was determined beforehand. This threshold represents the radius around a representative sample in whose range a newly added sample is classified as machine printed. If a sample is not in any radius it is classified as a handwritten character. To obtain the optimal local threshold, the radius of each representative machine printed sample is increased, until the local machine print precision reaches 98 %. The local machine print precision is defined as:

$$\frac{\#\text{machine printed samples in radius}}{\#\text{machine printed samples in radius} + \#\text{handwritten samples in radius}}$$

3.3 Existing Approach in smart FIX

The current approach for handwriting localisation in *smart FIX* uses a Support Vector Machine (SVM) to classify the CCs of a document into the classes handwritten / machine printed. This SVM was trained on CC samples from various handwritten as well as machine printed letters. In total, 21 manually designed CC features were created for the classification task, which can be grouped into three categories (see Table 3.1). The first group contains simple features like the width or height of the bounding box of a CC. The second group includes image moment features like the central moment or the orientation and eccentricity of the CC. For the features of the last group, the CC is rotated till $90°$ in steps of $0.1°$. For every step a bounding box will be drawn around the rotated CC and its area is calculated. The bounding box of with the minimal area of all steps will be used to compute the features of the last group (see Table 3.1).

From these three categories five features were selected by a rule-of-thumb, which considered classification performance as well as runtime. These five features (printed in bold, italic letters in Table 3.1) are used in the current version of *smart FIX*.

Tab. 3.1: Features for handwriting detection in *smart FIX*. Features printed in bold, italic letters are used in the current implementation.

Simple Features	
Width	w
Height	h
Height/Width Ratio	$\frac{h}{w}$
Area	$A = h \cdot w$
Black Area	A_{Black}
Black Area / Area Ratio	$\frac{A_{Black}}{A}$
Border Length	$\sum_{i=1}^{P} \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}$
Number of CC children	$\lvert CC_{Children} \rvert$
Compactness	$\frac{A_{Black}}{U}$
Image Moments	
M_{10}	$\sum_x \sum_y (x \cdot I(x,y))$
M_{01}	$\sum_x \sum_y (y \cdot I(x,y))$
$\boldsymbol{\mu_{11}}$	$\sum_x \sum_y (x - \overline{x}) \cdot (y - \overline{y}) \cdot I(x,y)$
μ_{02}	$\sum_x \sum_y (y - \overline{y})^2 \cdot I(x,y)$
$\mu 20$	$\sum_x \sum_y (x - \overline{x})^2 \cdot I(x,y)$
Orientation	$\Theta = \frac{1}{2} \arctan(\frac{2\mu_{11}}{\mu_{20} - \mu_{02}})$
Eccentricity	$\sqrt{\frac{(\mu_{20}+\mu_{02}) + \sqrt{(\mu_{20}-\mu_{02})^2 + 4 \cdot \mu_{11}^2}}{(\mu_{20}+\mu_{02}) - \sqrt{(\mu_{20}-\mu_{02})^2 + 4 \cdot \mu_{11}^2}}}$
rotated minimal Area	
Angle	$\Theta_{minArea}$
Area	$A_{minArea}$
Width	$w_{minArea}$
Height	$h_{minArea}$
Height/Width Ratio	$\frac{h_{minArea}}{w_{minArea}}$

4 Methodology

In this chapter the proposed approach of this thesis will be detailed, as well as the components of it. Section 4.2 outlines the different ways of extracting relevant regions on a document, which could contain handwritten annotations. In Section 4.3 the ANN architectures, which are used to determine whether an extracted region contains handwritten text or not, are explained.

4.1 Framework

The proposed approach consists of two exchangeable components, which are executed sequentially: An extractor and a classifier. A combination of both components is called locator, which performs the localisation of handwriting.

Fig. 4.1: Locator workflow

The first step for localising handwritten annotations on a document, is the extraction of image patches from the scanned document. For this task one of the four following sample extractors is used: *Sliding Window*, CC, *Quadtree*, and *Line Segment* (see Section 4.2). The extractors not only collect image patches in a different way, they also extract the spatial coordinates for each of the extracted patches.

In the second step an ANN, which was trained on the patches extracted by these extractor, classifies the extracted patch into the classes “contains handwritten text” or “does not contain handwritten text”. Three ANN architectures with different modifications were tested for this task: *AlexNet*, *SqueezeNet*, and a novel architecture called *HandDectNet*. Since the spatial positions of the extracted patches were saved by the extractor, it is now possible to determine in which region on the document handwritten text can be found.

4.2 Sample Extraction

The extraction of patches from the document plays an important role for the precision as well as for the runtime of the approach. Every classification of a patch takes

computational time, therefore it is necessary to minimise the number of extracted patches to only relevant ones. Relevant patches contain at least a certain amount of black pixels, otherwise it is likely that they will not contain any written text at all. It is also important that the extraction function will not skip any handwritten annotations. Thus, being too strict while sorting out irrelevant patches, will decrease the recall rate.

To reduce the search space of possible handwritten image patches, the sample extractor will only extract patches whose black / white ratio surpasses a given threshold t (0.005 by default). The black / white ratio, defined in Equation 4.1), is the ratio between the number of black pixels (the summation in the numerator) and the complete number of pixels in one patch. Function $I(x, y)$ denotes the pixel value at image position (x, y). Note that since the documents have been binarised, $I(x, y)$ can only evaluate to 0 for white pixels and 1 for black pixels. The default value for t was set to 0.005. It was chosen by a rule of thumb, taking into account the number of missed handwritten / machine printed patches and the number of retrieved irrelevant patches. When a patch surpasses the given threshold it has to be resized to 224×224. This is necessary due to the fixed input size of AlexNet and SqueezeNet.

$$\frac{\sum_x^w \sum_y^h I(x, y)}{w \cdot h} > t \tag{4.1}$$

In the following the different extraction methods developed for this thesis will be explained.

4.2.1 Sliding Window

This naive approach will extract patches from the original image by using a sliding window. To extract an image patch, a window of the size 224×224 is moved line-wise from the top left corner to the bottom right of the document image. The window can be moved not only in strides of the size of the window, but also in smaller amounts. This causes the extracted patches to overlap, which has two reason: firstly, it will increase the number of training samples, which will help to improve the accuracy of the neural net classifier. Secondly, during the classification it is less likely that an area which contains handwritten text is missed, since at least two patches will cover parts of this area. For this thesis, a relative small stride of 45 pixels was chosen.

4.2.2 Connected Components

Connected Components (CCs) are black pixels which are in direct neighbourhood of each other on an image and together form one big component. When extracted from a document, one can see that CCs of machine printed text often represent single letters or punctuation marks, whereas in handwritten text CCs are words.

With the help of *smart FIX* all relevant CCs can be easily extracted from a document. Some CCs like stamps or punctuation marks, which are usually not part of handwritten text, are filtered out by with the help of *smart FIX*. If the extracted CC is wider than 224 pixels, it will be sliced into patches of this width. To obtain the actual patch, the extracted CCs get centred in a 224×224 image.

4.2.3 Quadtree

Large parts of a document often contain only white background pixels. An extraction mechanism, which searches for relevant document parts should consider this and abort the search as soon as it is obvious that in a given region no more relevant patches are likely to be found.

The *Quadtree* extractor is an embodiment of this idea. It will search for non-white areas in the document image by dividing it into four equal parts. If the black pixel ratio (see Equation 4.1) of an image part is less than $t = 0.01$, it will not be further investigated. Parts which surpass this threshold will again be divided into four equally sized parts and the procedure will be repeated recursively. This is done until the patch width and height both are smaller or equal to 224. Like the CC extractor patches, they are centred in a 224×224 image.

4.2.4 Line Segments

Handwritten as well as machine printed text is usually written in horizontal lines and not in a vertical or diagonal orientation. This can be exploited for the sample extraction by using the line segmentation functionality of *smart FIX*, which recognises and extracts line segments from the document. As input samples for the neural networks, it is necessary to obtain 224×224 patches from the extracted lines. If a line segment is to big, a sliding window approach is used to split it into image patches smaller or equal to 224×224. In order to do so a window size of 224×224 and a stride of 224 is used.

4.3 Neural Net Architectures

During the work with *AlexNet* for this thesis, it became evident that the classification time presents a major challenge for the application of deep CNNs in *smart FIX*. The complete analysis process of one document in *smart FIX* should take only one or two seconds. This is critical, especially for systems which have to deal with more than 10,000 documents per day. For the classification of a document, the standard *AlexNet* already needs 6 seconds and more. This is not an acceptable runtime for a production system with document throughput, mentioned above.

To deal with this problem, different modifications were made to *AlexNet* to improve its runtime. These changes will be explained in the following section. Additionally, a novel architecture *HandDectNet* is presented, which was designed with the runtime constrain in mind.

4.3.1 AlexNet

One way to improve the speed of an ANN is to reduce the number of edges. *AlexNet* uses three fully connected layers for the classification at the end of the network. The number of edges between those three layers consist of $4096 \cdot 4096 + 4096 \cdot 1000$ edges. Since the classification task of this thesis is less complex (two classes) than the one of ImageNet (1000 classes), it seems justified to reduce the number of fully connected layers. This was radically done by replacing all three fully connected layers by a single one with two outputs.

4.3.2 SqueezeNet

The use of SqueezeNet for the classification task was mainly motivated by its small number of edges. With fewer edges a better runtime performance can be achieved. The only modification to the architecture of SqueezeNet has been made to the number of feature maps of the last convolutional layer. Due to the 1000 classes of ImageNet, 1000 feature maps were used in the vanilla version of this architecture. For the classification task of this thesis, the number of feature maps was again reduced to two.

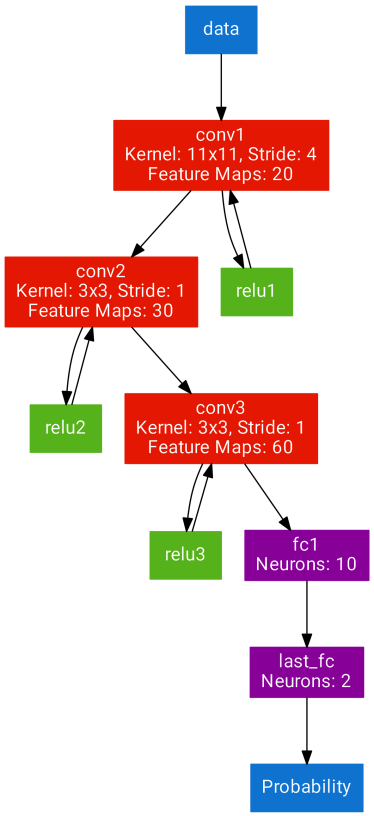
4.3.3 HandDectNet

AlexNet and SqueezeNet were both designed with the classification task of ImageNet in mind. Both are specialised to classify and to be trained on a huge data set with

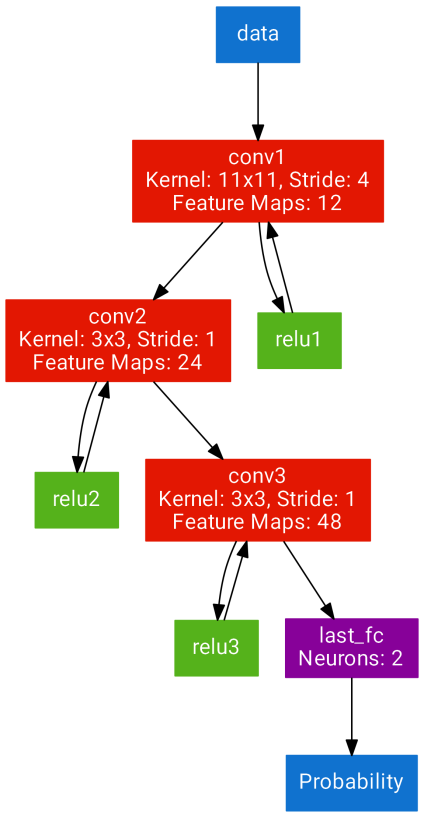
a great variety of visual information, like different object shapes or colours. The classification task of this thesis is much simpler. The visual information in the binarised document images are mostly restricted to geometric shapes of letters or lines. Since these kind of features are mostly learned by the first convolutional layers of an CNN (see Zeiler and Fergus [8]), a shallow net should fit this kind of classification task better. Instead of 1000, here it is sufficient to distinguish between two classes. Thus using more than one fully connected layer is unnecessary.

These considerations are embodied in the architecture of four versions of *HandDectNet*. In all four architectures the first three layer consist solely of three convolutional layers, each followed by a ReLu[1] layer. The last layers are fully connected layers, followed by a softmax layer. The difference between the four variants can be found in the number of feature maps per convolutional layer and in the number of fully connected layers at the end. *HandDectNet v1* is intended as a baseline, since it includes more feature maps per convolutional layer than any other *HandDectNet* version. Additionally, it has not one but two fully connected layers, with 10 and 2 neurons respectively (see Figure 4.2a). The idea behind this structure is to achieve the best classification performance in comparison to the other versions, but with the drawback of being slower. From *HandDectNet v1* to *v4* the goal of being as fast as possible was followed by continuously shrinking the net. Because the depth of a convolutional network is the key to its ability to build a feature hierarchy (see Section 2.2.2 as well as Zeiler and Fergus [8]), this meta parameter was not changed. The number of feature maps for each convolutional layer for the various versions of *HandDectNet* are shown in Figure 4.2. In comparison to *v2*, the feature maps of *v3* and *v4* are each reduced by the factor of 0.5 for each convolutional layer.

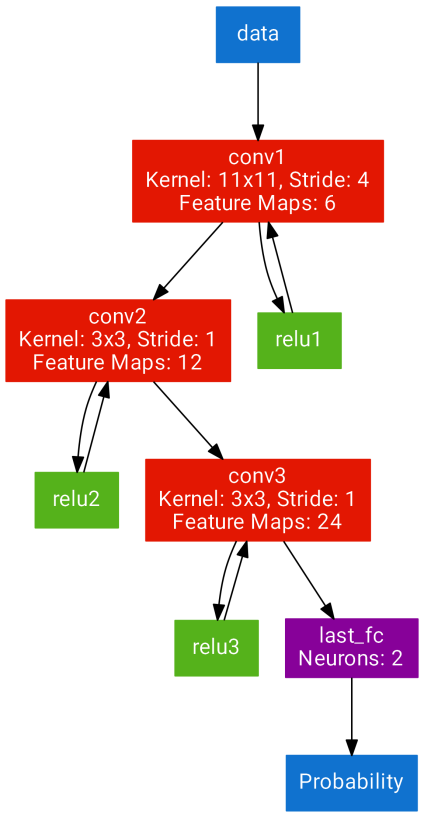
[1]The rectify activation function: $f(x) = max(0, x)$



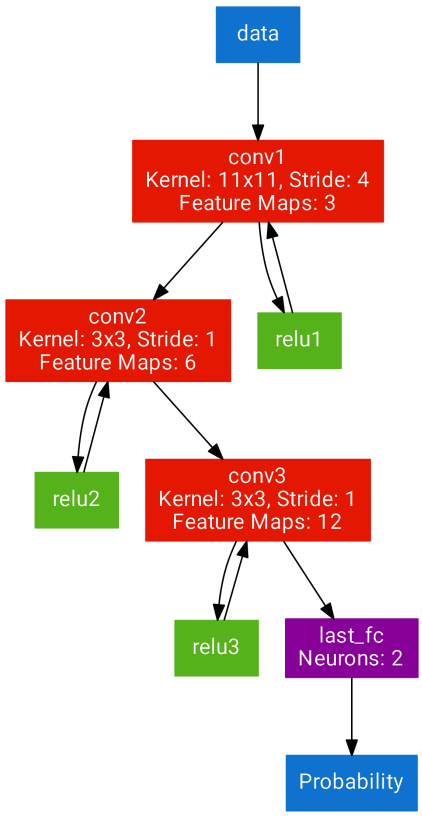
(a)v1



(b)v2



(c)v3



(d)v4

Fig. 4.2: HandDectNet Versions

5 Experiments

This chapter will detail the experiments, which were conducted to find the best combination of extraction method and ANN architecture. For all 28 combinations (see Figure 5.1) the performance and runtime were validated and tested on different data sets. The structure and creation of these data sets will be explained in Section 5.1. For measuring the performance of each combination different metrics were used, that are detailed in Section 5.2. Finally, the experiment setup will be outlined in Section 5.3.

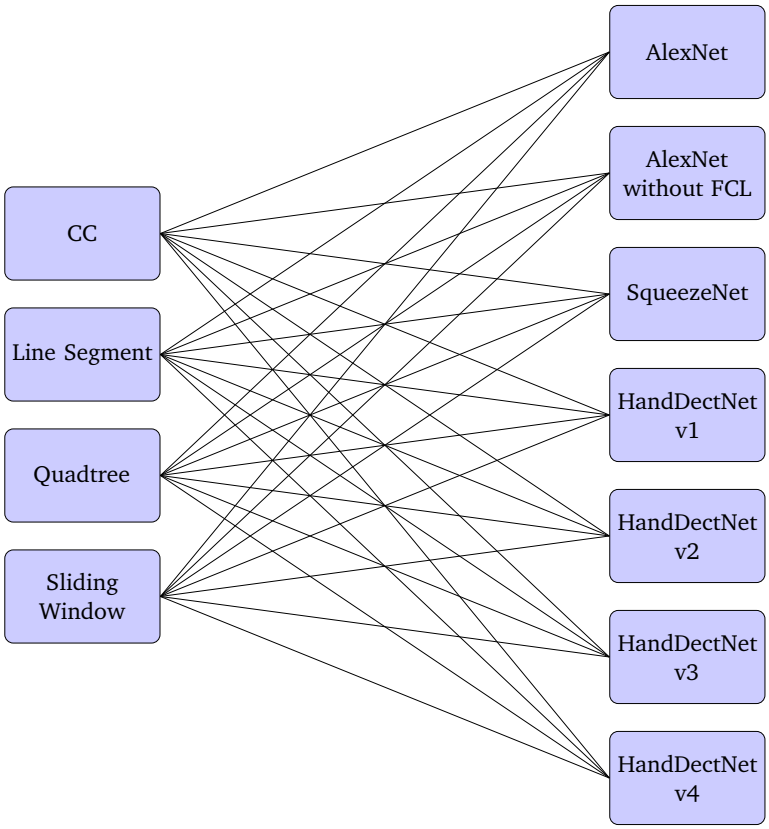


Fig. 5.1: Tested locators (extraction method / ANN architecture combinations)

5.1 Data

The data set used for training, validating, and testing consists of medical reports, prescription and other forms from health insurances. Many of the handwritten samples in this data set are taken from filled out forms, but also from handwritten

text annotating machine printed text. Another used data set contains handwritten letters from different writers.

All images were preprocessed by *smart FIX* with the functionality described in Section 2.4.2. The Ground Truth[1] (GT) was recorded from the preprocessed data set by hand using the *Verifier* from *smart FIX*. The GT consists of bounding boxes drawn around handwritten text on the images.

In the classification process, image patches of the document get extracted and classified. Because different extraction methods can produce different samples, all four extraction methods from Section 4.2 were used to generate four training / validation sets out of the main data set. The extracted patches of these four data sets have a size of 224×224. To assign the correct label to each of the extracted patches, the overlapping area of the patch and the bounding boxes from the GT for each document were determined. If and only if the overlap of a patch with a GT rectangle was greater than 50 %, it was labelled as handwritten. The threshold for the overlap was adjusted to 20 % for the *Sliding Window* and *Quadtree* extractors. Without this adjustment too many handwritten samples would have been missed during the extraction.

To prevent an imbalance of classes in the validation data sets, an equal amount of patches was randomly chosen from each of them. After that, 80 % of those patches were used to create the training set, whereas the remaining 20 % were used for the validation set. This split was done, such that both classes were equally represented in the training and validation set. The separation in training and validation set was done for all four data sets.

To compare the existing handwritten annotation detection with the new approaches, an additional test set was compiled. It consists of 49 randomly chosen images, which are not part of the data set used for generating the four training and validation data sets mentioned before. GT was recorded for this data set as well.

5.2 Metrics

The performance of the ANN classifiers was measured with different performance metrics. The quantities used in these metrics are subdivided into two groups: quantities used for the samples of the training / validation set and quantities used for the samples of the test set. This split is due to the fact, that both data sets are used to validate / test different characteristics of this thesis' approach.

[1] Set of image samples, which were labelled with their correct class.

Fig. 5.2: Test Quantities. Green: GT rectangle; Blue: predicted rectangle; Red: TP; Black: TN; Orange: FP; Yellow: FN

5.2.1 Validation Quantities

As mentioned in Section 5.1, the validation sets consist of different 224×224 image patches, randomly chosen from different documents. These patches are used to measure the classification performance of the different ANN architectures. Remember, every patch is labelled with "contains handwritten text" or "does not contain handwritten text", without any information where on the patch this class is present. Thus, this data set is only used for validating the classification performance and not the localisation performance.

The quantities represent the number of appearances of four different classification results during the validation of a classifier. Depending on the label of a patch (Actual Patch Class) and its predicted class (Predicted Patch Class), four different classification results for one patch can occur during the validation of a classifier. These are listed in Table 5.1.

Tab. 5.1: Validation Quantities

		Predicted Patch Class	
		Handwritten	None-Handwritten
Actual Patch Class	Handwritten	TP	FN
	None-Handwritten	FP	TN

5.2.2 Test Quantities

In contrast to the validation sets, where each sample is a 224×224 extracted patch from a document, the samples from the test set are complete document images. Each handwritten text is labelled by a rectangle enclosing it, showing not only its class, but also its location on the document. This spatial information allows us, to test the localisation ability of the thesis' approaches. For measuring the localisation ability of the locators, the definition of the quantities (Table 5.1) was slightly changed.

True Positives is the number of black pixels on a document, which are covered by an overlapping GT and predicted rectangle. C is the set of black pixels of a document predicted to be handwritten. G is the set of black pixels of a document labelled as handwritten in the GT.

$$TP = |C \cap G| \tag{5.1}$$

True Negatives is the number of black pixels on a document, which are not covered by any GT or predicted rectangle. N is the set of all black pixels in the document.

$$TN = |(N \setminus G) \cap (N \setminus C)| \tag{5.2}$$

False Positives is the number of black pixels on a document, which are covered by a predicted rectangle but not by a GT rectangle.

$$FP = |C \setminus G| \tag{5.3}$$

False Negatives is the number of black pixels on a document, which are covered by a GT rectangle but not by a predicted rectangle.

$$FN = |(N \setminus C) \setminus (N \setminus G)| \tag{5.4}$$

5.2.3 Performance Metrics

Based on the collected validation (Section 5.2.1) and test (Section 5.2.2) quantities, the following performance metrics can be determined.

Precision shows how many patches / black pixels were correctly classified as handwritten (True Positives (TP)), in comparison to all handwritten patches / black pixels. To put it another way: Precision expresses the ability of a classifier, to correctly predict the class of a handwritten patch / black pixel.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5.5}$$

Recall is the amount of patches /black pixels (TP) correctly recognised as handwritten, with respect to all patches / black pixels recognised as handwritten.

$$\text{Recall} = \frac{TP}{TP + TN} \tag{5.6}$$

F-measure is a metric combining precision and recall. It averages both values by using the harmonic mean. In this thesis, this metric is used to rank the classifiers and locators.

$$\text{F-measure} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.7}$$

Mean classification time Besides obtaining the classification performance, the mean classification time per document was also measured. Due to their different data set structure, this was done differently for the validation and test set.

The mean classification time during the validation is calculated by multiplying the average runtime per sample of classifier c, with the average number of samples per document $\varnothing N_e$ of validation set e. Because every validation set e consists of patches from different documents, $\varnothing N_e$ was determined beforehand. The average classification time of classifier c on validation set e is defined by Equation 5.8.

$$\varnothing T_{ce} = \varnothing t_c \cdot \varnothing N_e \tag{5.8}$$

The mean classification time on the test set is defined as the average runtime of a locator to extract and classify all patches on a document. N is the number of documents in the test set, which is 49 for this thesis. The function $t_{ce}(d)$ denotes the runtime of a classifier / extractor combination ce on document d.

$$\varnothing T_{ce} = \frac{\sum_d t_{ce}(d)}{N} \tag{5.9}$$

5.3 Setup

In the following, the experiment setup of training, validating and testing the different locators is detailed. For the first part of the experiment all ANN architectures were trained on all four training data sets. After each training, a *Python* script validated the classifier performance and runtime. It collected the validation quantities defined in Section 5.2.1 and computed the performance metrics mentioned in Section 5.2.3.

To summarise the results of each classifier on all validation sets, their performance results (precision, recall, runtime) were separately summed up and averaged over the number of validation sets (here 4). These summarised performance results can

be seen in Figure 6.3. Based on the averaged precision and recall, the F-measure value for each classifier was determined to rank their performance.

In the second part of the experiment, all 28 locators (classifier / extractor combinations) were tested on documents of the test set. This was achieved with the implementation of this thesis' approach in *smart FIX*. For every locator, the performance metrics mentioned in Section 5.2.3 were evaluated based on the collected test quantities defined in Section 5.2.2.

Similar to the validation set a summary of the classifier performances was compiled for the test set. This was achieved by summing up the performance results of the locators with the same classifier and averaging the summation over the number of extractors (here 4). Although the extractor does not take directly part in the classification process, it has an indirect influence by finding and pre-selecting the patches for the classifier. The influence of the extractor on the locator performance was measured indirectly, by grouping the locator results by their extractors. For each extractor the grouped performance results were summed up and averaged by the number of classifiers. Because all classifiers were used with every extractor, the differences in this grouped results can be explained by the different ways of extracting patches for the classifier.

6 Evaluation

In the following the observed runtime and performance results of the experiments will be analysed. The analysis will investigate the results of the classifiers, extractors, and locators. Section 6.1 analyses the results of the Top-5 locators on the test set and compares them to the performance of the baseline. An evaluation of the classifier performances on the validation set is done in Section 6.2. The last section studies the performance of the locators in dependency to their extractor.

6.1 Top-5 Classifier / Extraction Method Combinations

The Top-5 locators, tested on the test set and ordered by their F-measure value (see Equation 5.7), can be seen in Figure 6.1. With precision rates of 70 % and above, nearly all Top-5 locators reached very good results. The first and second best locator are using *AlexNet* and *AlexNet without FCL* as classifier. The best precision rate among all locators was also reached by both of them with 85 %. The third best locator, which is using the *HandDectNet v2* architecture, obtained a precision rate of 78 %.

As expected, the locators using *AlexNet* as classifier were much slower as the ones using *HandDectNet* classifiers. Even the *AlexNet without FCL* architecture only achieved a runtime of 6.76 seconds per document (see Table 6.2). Here it got outperformed by the *HandDectNet* classifiers, which needed at most 1.69 seconds per document.

Interestingly nearly all Top-5 locators are using the *Line Segment* extractor to gather patches for classification. A comparison between the different extraction methods and their impact on the classifier performance and runtime will be considered later in Section 6.3.

Compared to the baseline, nearly all Top-5 locators achieved equivalent or better performance results. Especially the *Line Segment / AlexNet* locator performed very well, since its precision rate is 15 % better than the baseline. In terms of speed, all *HandDectNet* classifiers reached classification times equivalent to the baseline. While *HandDectNet v2* was only 0.7 seconds slower, *HandDectNet v4* showed the same runtime as the baseline. It has to be considered, that the performance comparison with the baseline has to be taken with a grain of salt. The SVM was trained on a

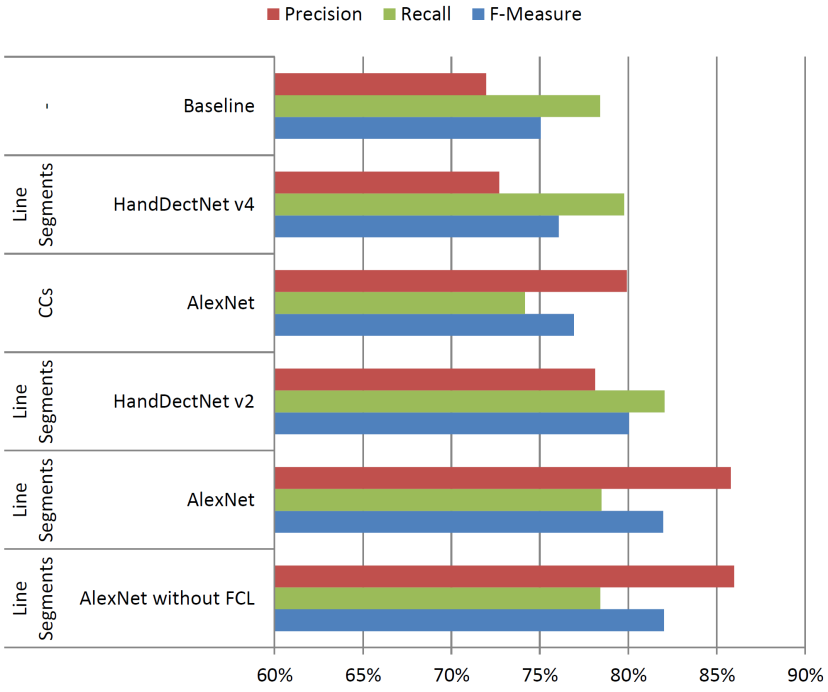


Fig. 6.1: Top-5 locators, ascending ordered by their F-measure value from top to bottom.

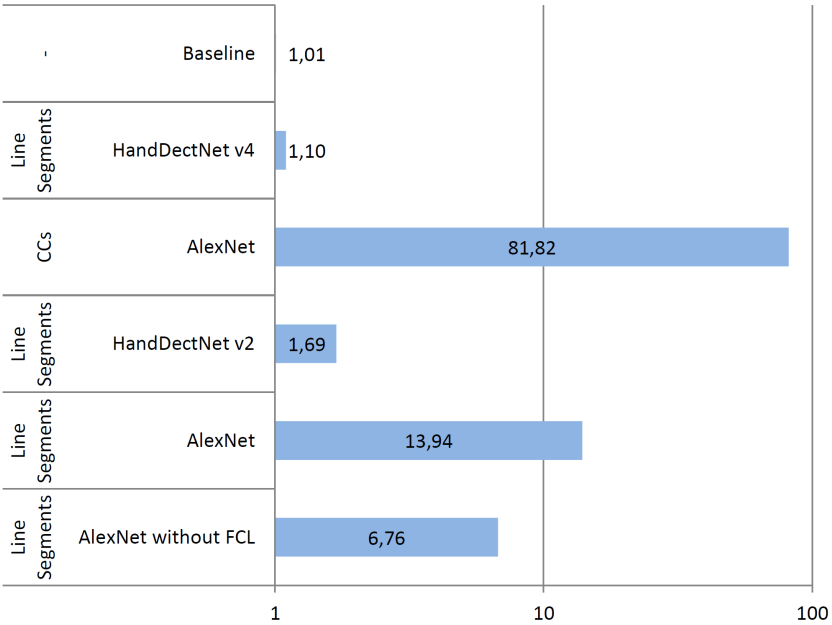


Fig. 6.2: Top-5 locators runtime. The unit of the results is seconds. Note that the axis is scaled logarithmic to the base of 10.

subset of the training set which was used for all classifiers. This could result in a lack of sample variety, leading to the inferior performance results of the baseline. To achieve a true performance comparison, the SVM has to be trained with the same training samples as the ones used to train the ANN classifiers.

6.2 Classifier

Figure 6.3 shows, nearly on all validation sets *SqueezeNet* and *AlexNet* surpassed the *HandDectNet* classifiers regarding their F-measure scores. Nevertheless, *HandDectNet v2* and *HandDectNet v3* reached very good results, both with F-measure scores higher or equal to 90 %.
In terms of runtime, the *HandDectNet* classifiers have been faster in comparison to the *SqueezeNet* and *AlexNet* architectures (see Table 6.4). In most of the cases the average classification time for a document on the validation sets took 4.89 to 14.55 seconds. This outperforms the other ANN architectures, which needed 37.93 to 180.74 seconds per document, by a large margin. Here it is important to note, that the classification time on the validation set is much higher compared to the runtime on the test set, because the classification process was done via a *Python* script. In contrast to this, the test set evaluation was done via *smart FIX*, which is much faster due to its C++ implementation.

AlexNet and *AlexNet without FCL* were the best performing classifiers on all four validation sets. Both achieved nearly identical performance values of 94 % and above (see Figure 6.3). In comparison to these two architectures, *SqueezeNet* reached nearly the same results since it performed only 1 % worse at most.
With a runtime of 180.74 seconds, *AlexNet* is the slowest of all classifiers on the validation sets. One reason for this, like mentioned in Section 4.3.1, is the high number of edges between the last three fully connected layers. By replacing them with one fully connected layer with only two outputs (*AlexNet without FCL*), the runtime decreased by a large margin of 142.81 seconds. Similar to *AlexNet without FCL*, due to its compact network architecture, *SqueezeNet* performed with 43.63 seconds much better.

Among all *HandDectNet* versions, *v2* and *v3* reached the best performance values. Their results are very similar, with only ~1 % difference in their F-measure value. Interestingly *HandDectNet v1* and *v4* showed worse results, with F-measure values smaller than 12 % compared to *v3*. Their relative poor performance could be explained by their too complex (*v1*) or too simple (*v4*) architecture. It seems that a balance between those two extremes works best for the classification task of this thesis.

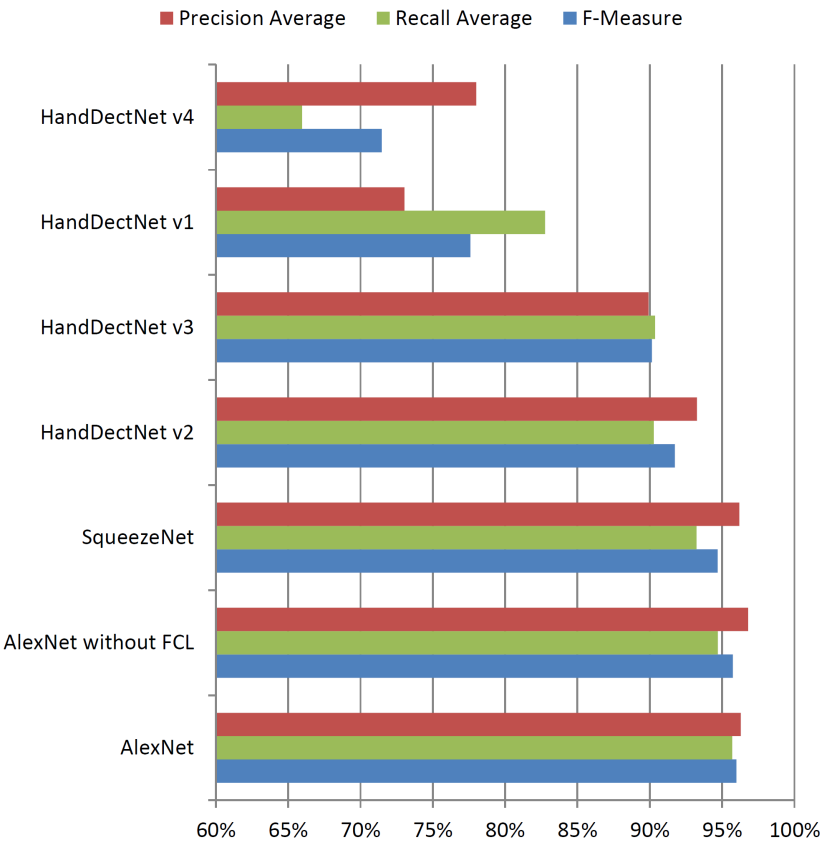


Fig. 6.3: Classifier performance summary on the validation sets. The classifiers are ascending ordered from top to bottom by their F-measure value.

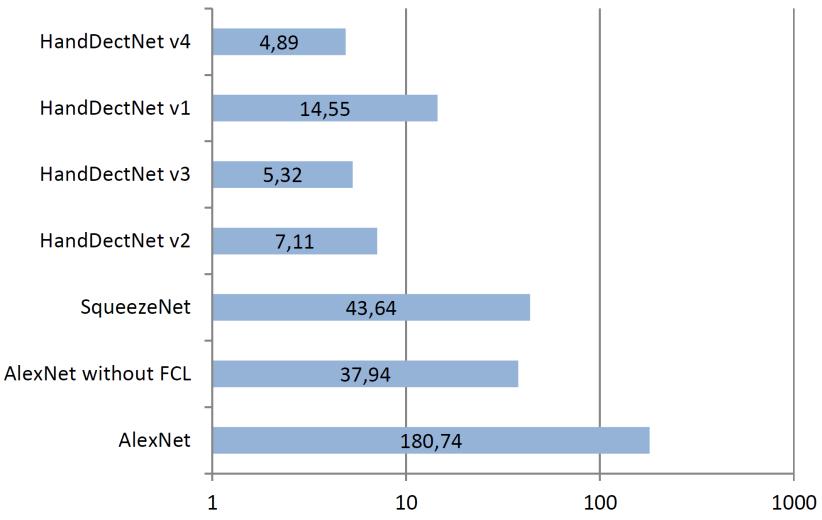


Fig. 6.4: Classifier runtime summary on the validation sets. The unit of the results is seconds. Note that the axis is logarithmically scaled to the base of 10.

Note, that the classification performance results of all ANN classifiers are slightly flawed, because a few none-handwritten samples were wrongly labelled as handwritten in the GT. This was probably caused by an alignment problem between the GT rectangles and the corresponding document during the automatic creation of the GT samples (see Section 5.1). Obviously it can be assumed, that a correction of the incorrect labels will lead to an increase in the performance of all ANN classifiers.

6.3 Extraction Methods

As mentioned in Section 5.3, the performance of the extractors is measured indirectly by the results of the locators on the test set. The performance results, averaged and grouped by each extractor, can be found in Figure 6.5.

It can be seen, that locators, which use the *Sliding Window* or *Quadtree* extractor perform significant worse, than locators using the *CC* or *Line Segment* extractor. When comparing the extracted patches of the four extractors, *Sliding Window* and *Quadtree* samples show a common pattern. A few of these samples were extracted from parts of the document images, where only the gap between two text lines is visible (see Figure 6.8b). Whereas the unimportant white gap takes up the most space on the patch, the more important text lines are only partly visible. For the classification, these lines contain valuable information enabling the classifier to distinguish between the two classes. This problem could be reduced for the *Sliding Window* extractor by choosing a smaller window stride, leading to an higher overlap between the extracted patches. With an higher overlap, the likelihood to extract these bad samples in a particular region will decrease. Due to time constrains, experiments with different window strides were not conducted but can be part of potential future work.

Compared to the other extractors, *Quadtree* shows the worst recall rate. This can be explained by its way of extracting patches from the document. Like explained in Section 4.2.3, the *Quadtree* extractor will split the document in four equally sized parts. If one of the four parts does not contain enough black pixels to pass the threshold, it will not be further investigated even if handwritten text is present in this quadrant. Many potential handwritten samples located in this region of the document will be missed.

Locators with the *Sliding Window* or *Quadtree* extractor have a much shorter classification time per document, compared to locators using the *Line Segment* or *CC* extractor. This can be explained with the relative low number of extracted samples, which is on average 100 patches per document for the *Quadtree* and 105 patches

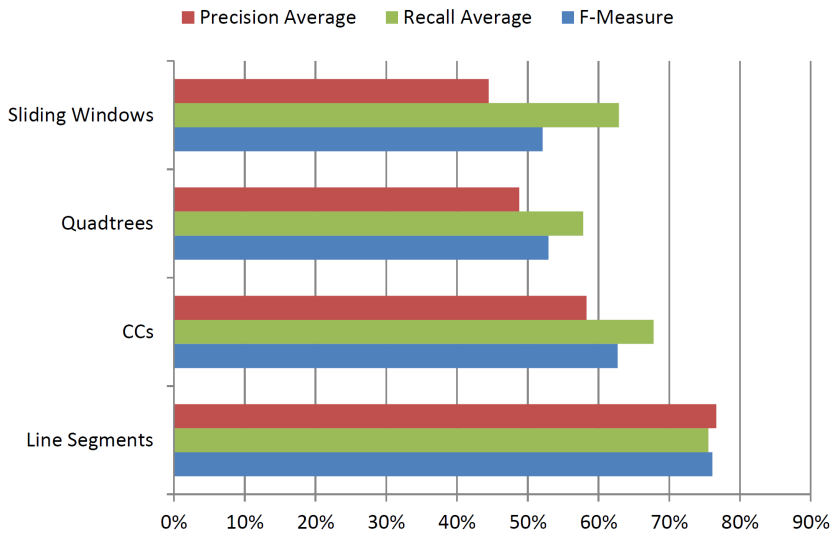


Fig. 6.5: Extractor Performance Summary on the test set. The extractors are ascending ordered from top to bottom by their F-measure value.

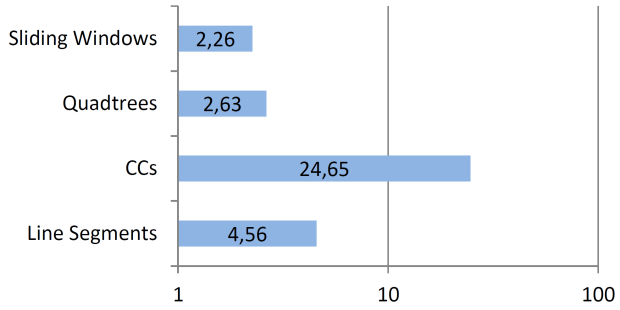
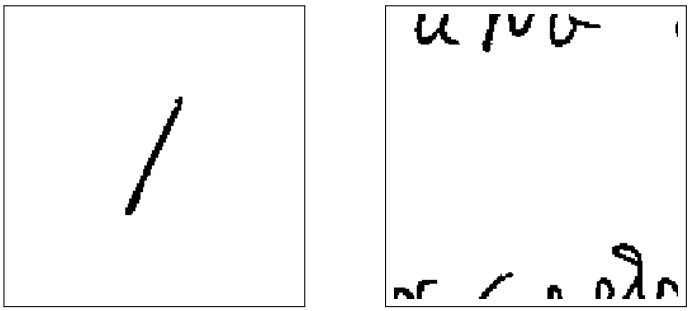


Fig. 6.6: Extractor Runtime Summary on the test set. The unit of the results is seconds. Note that the axis is logarithmically scaled to the base of 10.

Fig. 6.7: Bad Samples



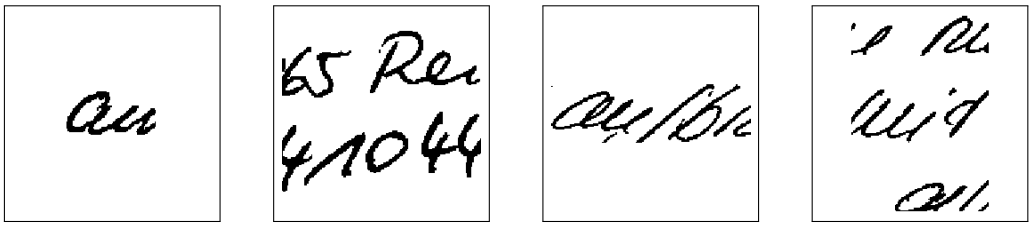
(a)CC **(b)**Sliding Window

per document for the *Sliding Window* extractor. In contrast to this, the *Line Segment* extractor extracts on average 148 patches, whereas the *CC* extractor extracts even 409. This relative low number of samples can be explained by the aforementioned problems of the *Quadtree* and *Sliding Window* extractor. The *Quadtree* extractor seems to sort out patches too early in the extraction process. The *Sliding Window* extractor produces samples like in Figure 6.8b, which get sorted out because of their high portion of white pixels.

Better results have been achieved by the *CC* and *Line Segment* extractor. They extract characters or text lines directly from the document, which reduces the likelihood of extracting irrelevant patches. Besides of the reduction of irrelevant patches, both extractors will also produce more valuable samples for training and classification. The content of their samples consist mainly of handwritten or machine printed text, which is exactly the information a classifier needs to distinguish between those two classes. Additionally, both methods centre the extracted information in the middle of the patch, which further eases the training and classification.

With an average precision rate of 76 % on the test set, the *Line Segment* extractor performs better than the *CC* extractor with a precision rate of 58 %. A possible reason for the difference are CCs, which barely contain any information, like strokes or other similar small CCs (see Figure 6.8a). A distinction between handwritten and none-handwritten samples is often not possible with such samples. Patches extracted by the *Line Segment* extractor do not suffer from this problem. Most of the time CC samples contain only one character, whereas *Line Segment* samples contain a complete line of characters. This increases the amount of information available in one patch. Since a line of text can be seen as a group of semantically related characters, these samples also offer a lot more context to the classifier than a single CC sample. The samples with the highest confidence among each validation set are shown in Figure 6.9 (here exemplary for *HandDectNet v3*). These samples seem to support this hypothesis. Especially the examples from the *Sliding Window* and

Fig. 6.9: Best TP samples from *HandDectNet v3*



CC | Sliding Window | Line Segment | Quadtree

Fig. 6.10: Best TN samples from *HandDectNet v3*

CC | Sliding Window | Line Segment | Quadtree

Quadtree extractor make it clear, that more given context yield better classification results.

If compared in terms of runtime it is obvious, that the *CC* extractor is by far the slowest extractor, with 24.64 seconds per document (see Figure 6.6). Its long runtime can easily be explained by its large number of extracted samples, which are on average 409 patches per document. The *Quadtree* and *Sliding Window* extractor are, with around 2.44 seconds, twice as fast as the *Line Segment* extractor (4.56 seconds), but also have much worse classification results.

7 Conclusion

In this thesis a novel Deep Learning approach for the localisation of handwritten annotations in business documents has been proposed. It utilises different sample extraction methods in combinations with well known neural network architectures like *AlexNet* or *SqueezeNet* for classification.

SqueezeNet and *AlexNet* showed nearly equivalent performance results for the classification task. Both reached excellent performance results, with an average precision rate of 96 % and an average recall rate from 93 % up to 95 % on the validation sets. Since one of the objectives of this thesis was to apply the studied approaches to *smart FIX*, their runtime has been studied as well. The experimental results showed that, due to its more complex structure, *AlexNet* had an inferior runtime in comparison to *SqueezeNet*. It was shown that removing the last two fully connected layers from *AlexNet* did not impact its performance for the classification task of this thesis. But its runtime decreased from 180.74 seconds to 37.93 seconds per document, which makes it comparable to the runtime of *SqueezeNet*. It can be concluded that *SqueezeNet* as well as *AlexNet without FCL* are well suited for classification tasks like this and can be served as a baseline for similar future tasks.

Even though *AlexNet without FCL* and *SqueezeNet* showed good performance results and a better runtime as *AlexNet*, they are too slow for being used in the productive context of *smart FIX*. To meet the runtime requirements of *smart FIX*, the novel CNN architecture *HandDectNet* was designed, with the runtime constrains of *smart FIX* in mind. Four different versions of *HandDectNet* were tested for their classification performance and runtime. It could be shown that *HandDectNet v2* in combination with the *Line Segment* extractor exceeded the performance results of the baseline[1] on the test set. With a precision of 78 % and a recall of 82 % it outperformed the baseline, while at the same time reaching a comparable runtime of 1.67 seconds per document.

Furthermore, four different approaches to extract image patches for classification have been studied and their performance and runtime was compared. According to the experiments, the *Sliding Window* and *Quadtree* extractors were outperformed

[1] Please take into account that the comparison of the performance results between the baseline and the locators is questionable, because the SVM used in the baseline was only trained on a subset of the available training samples.

by the *CC* and *Line Segment* extractor. Their superior results can be attributed to their ability to directly extract those parts of the document which contain text. Since the *CC* extractor directly extracts single characters, one could assume that it yields the most accurate localisation results. As the experiments show, this assumption can not be confirmed. Handwritten CCs patches are often difficult to distinguish from machine printed CCs. Furthermore, even samples within the same class are hard to distinguish. This seems to make the training of the classifiers more difficult. In contrast to the CC samples, the samples extracted by the *Line Segment* extractor deliver more context to the classifier. Instead of one character a complete line of characters is used by the classifiers to distinguish between handwritten and machine printed text. With a precision rate of 76 % and a recall rate of 75 %, the *Line Segment* extractor showed the best performance results on the test set. Considering these results, it is no surprise that the *Line Segment* extractor is also present in four of the Top-5 locators.

Considering performance as well as runtime, the best extractor / classifier combination is the *Line Segment* extractor together with *HandDectNet v2*. This locator is a compromise between localisation performance and runtime. Even though the combination *Line Segment / AlexNet without FCL* achieved a better precision rate (85 % vs. 78 %), due to its runtime of 6.76 seconds per document, it is not practical usable.

The approach of this thesis could be further improved by investigating into the well performing *HandDectNet v2* and *v3* architectures with different visualisation methods used by Zeiler and Fergus [8]. This would contribute to a better understanding of how these networks learn for this specific task and how they can be improved.
Since the focus of this work was on the evaluation of different approaches instead of a single one, a more detailed investigation of a few selected classifiers can be conducted in possible future works. For example, one can expect that the performance of the Top-5 classifiers can be easily increased by experimenting with the values of various meta-parameters like the learning rate or the maximal number of training iterations.
Another simple way to increase the performance of the classifiers could be to correct the incorrectly labelled GT samples, mentioned at the end of Section 6.2. Once the GT is corrected, a retraining of all classifiers will likely result in a performance improvement for all of them.
The runtime could also be decreased by implementing different methods for filtering samples before the classification. One approach for filtering line segments could be realised by collecting statistics over the typical height of machine printed text lines and sorting them out before any time consuming classification has even started.

Bibliography

[1] Kjell Magne Fauske. *Example: Neural network*. 2006. URL: http://www.texample.net/tikz/examples/neural-network/ (visited on Nov. 23, 2016) (cit. on p. 7).

[2] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, et al. „SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size“. In: *arXiv:1602.07360* (2016) (cit. on pp. 10–13).

[3] E. Kavallieratou and S. Stamatatos. „Discrimination of machine-printed from handwritten text using simple structural characteristics“. In: *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.* Institute of Electrical & Electronics Engineers (IEEE), 2004 (cit. on p. 15).

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. „Imagenet classification with deep convolutional neural networks“. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105 (cit. on p. 10).

[5] Samuel J. Pinson and William A. Barrett. „Connected Component Level Discrimination of Handwritten and Machine-Printed Text Using Eigenfaces“. In: *2011 International Conference on Document Analysis and Recognition*. Institute of Electrical & Electronics Engineers (IEEE), 2011 (cit. on p. 16).

[6] Michael Plotke. *3D Image-Kernel Convolution Animation*. 2013. URL: https://commons.wikimedia.org/wiki/File:3D_Convolution_Animation.gif (visited on Oct. 17, 2016) (cit. on p. 9).

[7] Matthew Turk and Alex Pentland. „Eigenfaces for Recognition“. In: *J. Cognitive Neuroscience* 3.1 (Jan. 1991), pp. 71–86 (cit. on p. 16).

[8] Matthew D. Zeiler and Rob Fergus. „Visualizing and Understanding Convolutional Networks“. In: *CoRR* abs/1311.2901 (2013) (cit. on pp. 23, 40).

List of Figures

List of Tables

Declaration

I, Dominik Bermühler, declare that this thesis titled, „Application of Deep Learning Techniques in a Commercial Document Analysis System“ and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Kaiserslautern, April 18, 2017

Dominik Bermühler