UPPSALA
UNIVERSITET

Independent Project in Information Engineering
May 2016

Automatic reading and interpretation of paper invoices

ADC invoice

Boström, Carl
Herelius, Johan
Hugosson, Mathias
Maleev, Sergej

Computer and Information Engineering Programme

Civilingenjörsprogrammet i informationsteknologi

UPPSALA
UNIVERSITET

Institutionen för informationsteknologi

Besöksadress:
ITC, Polacksbacken
Lägerhyddsvägen 2

Postadress:
Box 337
751 05 Uppsala

Hemsida:
http:/www.it.uu.se

Abstract

Automatic reading and interpretation of paper invoices

ADC invoice

Boström, Carl
Herelius, Johan
Hugosson, Mathias
Maleev, Sergej

Manually typing long numbers on paper invoices is tedious and time consuming work. The task of typing all the fields of an invoice into a text editor was given to two subjects working with bookkeeping, and the average time consumed was measured to be five minutes. The time and cost spent on manual typing will accumulate for companies that receive a lot of invoices. Swedbank [1] along with other banks, have addressed this issue with a mobile application that reads and interprets the numbers on an invoice using the built in camera. This solution is directed to the public and the extracted information cannot be imported into bookkeeping software. A standalone software for digital reading and interpretation of scanned invoices is our solution for companies in regards to this issue.

There is already technology available that will interpret written text. These techniques were applied in our work, but the focus of this project was to implement algorithms for finding the location for a specific number and resolve what bookkepping terms the number references e.g. OCR, IBAN numbers, etc. This hampered by the missing of a general invoice layout.
52% of all the sought information was extracted correctly and almost 95% of the bookkeeping details that are changed most frequently on invoices from the same service providers were extracted correctly. The average time it takes for the application to extract the vital data is 30-40 seconds.

Extern handledare: Håkan Lundmark, Crowderia AB
Handledare: Sofia Cassel och Björn Victor
Examinator: Björn Victor

Sammanfattning

Manuell inmatning av alla långa nummer på en pappersfaktura är ett tråkigt och tidskrävande arbete. Uppgiften att skriva in alla fält från en faktura till en ordbehandlare gavs till några personer som jobbar med bokföring. Tiden mättes till ett snitt på 5 minuter. Tiden det tar och den kostnad det medför kommer att ackumuleras för företag som får regelbundna fakturor från sina leverantörer. Swedbank [1], och andra banker, har adresserat problemet med en mobil applikation som kan läsa och tolka nummer på en faktura via mobilkameran. Den här lösningen riktar sig mot privatpersoner och den extraherade informationen skickas direkt till banken och kan inte importeras i ett bokföringsprogram. Ett fristående program för digital läsning och tolkning av inskannade fakturor är vår lösning till företagen för detta problem.

Det finns redan väldigt kraftfull teknologi för att tolka skriven text. Dessa tekniker användes i vårt arbete, men svårigheten i detta projekt var att implementera algoritmer för att lokalisera ett nummer och lista ut vilken bokförings referens det representerar, t.ex. OCR, IBAN etc. Detta försvåras av avsaknaden av ett generellt faktura format.

52% av den önskade information var korrekt extraherad och nästan 95% av bokföringsdetaljerna som ändras mest mellan fakturor med samma avsändare extraherades korrekt. Tiden det tog var i genomsnitt för programmet att extrahera informationen var vid dessa försök 30-40 sekunder.

Contents

1 Introduction

In a world where you can buy almost anything and have the bill sent to you later, a lot of paper invoices can accumulate. Reading and organizing all the information an invoice contains is pretty dull work, not to mention time-consuming. This project attempts an analogue to digital conversion of paper invoices to automate the process of typing analogue information into a digital bookkeeping system. The product is an application that uses *OCR, Optical Character Recognition*, to extract vital data from a scanned image of an invoice. OCR is a well-established concept in the field of pattern recognition. Considering the wide amount of OCR engines that are already available, we want to use an existing *API, Application Programming Interface,* with OCR-support to create specialized software that can recognize invoice content.

The challenge we face involves how to identify blocks of information that are intended to be interpreted by a human. The context of the text might be obvious to most people, but is it possible to write an algorithm that captures that context? In either case we will attempt a solution that can at least aid the efforts of invoice related work.

2 Background

Companies, government institutions and private individuals all have in common that they all have to pay service providers. Companies and government institutions also have to do bookkeeping of their expenses. If the service is not paid directly at the scene, there are three major options available; paper invoices, electronic invoices and direct debit. Direct debit means that the customer gives the service provider the privilege to draw the amount from the customer's bank-account by pre-approving such transactions, making it harder for the customer to have control over the payments.

Electronic invoices. This is an environmentally friendly, cheap and efficient way of paying digitally. The customer gets the invoice digitally instead of on paper. The invoice is easily imported to online bank services and bookkeeping software. Visma, a major company in the field of bookkeeping and invoice handling, has written an article [2] where they describe the advantages of electronic bookkeeping, claiming that it is 15 times faster to handle electronic invoices than those on paper.

Paper invoice. But what if the service provider cannot offer electronic invoices, even though more and more service providers do so [3]? If the only remaining choice is paper invoices, then we have a conflict between the analogue invoice and the digital bookkeeping software that connects to online bank services. Manually typing all the information on an invoice is a tedious work that will take around five minutes for each invoice, which was discovered when people working with bookkeeping were used as research for this project. We find this issue being addressed in multiple solutions, such as bank applications and bookkeeping software, some of them without support for importing the data into other programs and some that sends the invoice information to another company for digitizing which will harm the business integrity.

Character recognition. Reading and interpreting characters and objects has been researched for a long time. The first machines that were able to recognize characters were developed in the 1950's, however such technology was only recently made available for commercial use. [4] Modern optical character recognition technology can recognize analogue text with impressive accuracy, but the nature of the text is context-sensitive, thus a more specialized solution could potentially improve both efficiency and accuracy. However, an application that makes occasional errors would have to be supplemented by manual processing.

The client. This software development project is supported by Crowderia [5], an IT company founded in 2003. Crowderia provides IT-tools for process management, product management and logistics for a diverse group of companies. The project was conceived in dialogue with our contacts at Crowderia who were able to provide us with feature suggestions and help with certain development decisions regarding invoice layouts. Crowderia's primary intention has been to apply our software solution onto their own business and potentially distribute the final product commercially by providing it to their customers.

3 Purpose, goals and motivation for the project

Purpose. The purpose of our application, ADC Invoice, is to reduce the workload for the employees who handle invoices for a company. The task of reading the information on invoices and typing it into a computer is time-consuming and very repetitive. We believe this task is better left for semi-automated systems. ADC Invoice provides a means to process invoices faster, the idea is that a computer can easily type payment and contact information faster than any human, and given the repetitive nature of the task it is ideal to be executed with software.

Project goals. A digital copy of an invoice shall contain a standardized set of fields that make out the invoice, e.g. organization number, payment date and invoice number. We have attempted to digitize these fields according to what is deemed important. The selected content of the digital translation has been subject to change during development to make sure the information is useful.

The digital copy is to contain the unprocessed image as well as the extracted data for bookkeeping and historical purposes. The application shall:

- Take a scanned copy of an invoice as input.
- Extract the vital data, like addresses and payment details, from input.
- Generate a template for outgoing payments.
- Generate a template for bookkeeping.
- Allow support for bookkeeping software such as Visma or Fortnox.
- Offer a mode for human authorization as the correctness of each case can not be guaranteed.

This way the user can complete the digitization process with the following steps.

1. Scan the invoice.
2. Load an image of the invoice into the application.
3. Authorize the outgoing payment or send the necessary data to whoever does the authorization.

Motivation. While the field of optical character recognition offers applications that are able to perform general text recognition, we could not find a single standalone-option specialized on invoices. More specifically there appears to be no available applications that reads, recognizes and handles the data being read on the invoice. Our project differs in that our application will help the user save time by not having to either copy-paste the data from another OCR-application or by typing everything in manually.

Having access to semi-automated invoice handling allows a company to spend fewer man-hours on this task, thus the company can work more effectively. A possible side-effect is that the company will require fewer employees to handle their workload, which on an individual scale could in fact be negative. However for the company as a whole, productivity can be increased, thus increasing its worth.

Deliminations. Considering the workflow of invoice digitization, being able to use a smartphone application could be highly practical. However this would require substantial work efforts in smartphone development, thus we decided to limit the project to only the desktop application.

4 Related work

Our application is based on two major concepts. One is using OCR for reading machine-written text on paper and converting it to digital text. The other is the idea using computers to automate the task of extracting and sorting through the data in invoices. The former offers a broad selection of applications and devices ranging from reading a single word and translating it to the more advanced image recognition software that can even interpret facial expressions. The latter is more specialized and offers features such as automatic insertion of data into a given *ERP, Enterprise Resource Planning*, a solution for automatic payment and storage into specific bookkeeping software.

4.1 Solutions for paper invoices

Lexmark's Oracle Invoice Processing. One solution that is already available with major similarities to ours is an application offered by Lexmark, a company [6] specializing in high-end IT-solutions for established companies. The oracle invoice processing solution [7] offers automatic scanning, data extraction, filing and payment of invoices. Which is what we were planning and more, however this requires you to integrate the software with oracles E-business suite [8]. Lexmark also offers support for other business suites and ERPs, boasting a network of partners in *FPA, Financial Process Automation,* from over 70 countries. One major drawback with Lexmark's solution is that it does require integration with the end-users business suite. If that is not possible then Lexmark cannot offer automatic invoice processing for said end-user. Our application differs in this aspect by being standalone, while it is supposed to be able to communicate with a given business suite it does not attempt to integrate with the business software but rather attempts to tag and sort the information on the invoice and present it in such a way that the business suite can read it right away. Preferably by passing a file with a specific convention for tagging and sorting. Another difference between ours and Lexmark's solution is the ability to handle invoices that were sent electronically. Our application only handles invoices that are scanned and imported to the application. The tagging and interpretation of the data is closely linked with the OCR section of our application.

Fortnox. Fortnox provides a service named ‘Fakturatolkning’ [9], which translates roughly to ’invoice interpretation’. The end-users of this service send invoices they want to have digitized to Fortnox. Once the invoice has arrived, physically or electronically, Fortnox applies their software solution onto the invoice to extract the vital information needed for bookkeeping from the invoice. The extracted information is validated by an employee at Fortnox after which it is inserted into the bookkeeping service offered by Fortnox. Much like our approach Fortnox choose to store an image of the original invoice paired with the digitized copy. As of today the current price for this service is 4.90 or 9.90 SEK, Swedish Krona, per invoice where the lower price is assuming the invoice is delivered electronically to Fortnox. The major difference between our approach and Fortnox’s is that the actual processing is done at a third party company, in this case Fortnox. Which leaves the possibility of a company renting this service having little to no idea of what products and services they’ve actually paid for. If there is a service that offers a one-time payment solution, then a service that is priced proportionally to the amount of invoices received will be undesirable for a load big enough. Our application could be offered as such an in-house solution that is locally deployed.

4.2 Applications for OCR

The field of Optical Character Recognition has been researched for many years and a lot of different applications have been developed. Today many Office applications provide OCR, e.g. Microsoft Office OneNote 2007 or FreeOCR. [10] These particular applications are very general and simply convert characters without regard to the context of the information. Our application also focuses on interpreting the information in order to be useful to the user.

Swedbank. Similar to ADC Invoice is Swedbank's application [1] that scans a number from an invoice and figures out what kind of information it represents. The problem here is that the user has to locate the information needed on the paper and hold the camera steady over the paper at a proper height in order for the OCR to work. Another issue is that the information is sent directly to the bank and can not be used in bookkeeping software. Swedbank's application appears to be designed for private users, who pay a few bills each month, while our application is intended for corporate usage.

Voice4u TTS. Voice4u [11] provides an application that combines OCR with a *TTS, Text to Speech,* engine. The user takes a photo of a text they want to read, an article in a paper for example, and the application then digitizes the text and sends it to the TTS which reads the text out loud. When trying it on text written in Swedish, the characters Å and Ä were replaced with A and Ö with O. Thus it appears the OCR engine that was used only supports ASCII characters. Nevertheless this application can be of great use to people with visual impairment or other reasons for difficulty in reading.

Google Translate. Google offers an application [12] that is designed for on the fly translation of text you can capture with a mobile device. One interesting use for the application is translating signs you come across as seen in fig. 1 where an exit sign has been translated to the corresponding Swedish word 'utgång'. Upon testing the application it appears that smaller texts of a single or a few words can be translated locally in the device. Whereas longer texts such as articles require an internet connection. Since google translate is a general OCR application, it does not recognize an invoice to specifically contain invoice information. We attempt to offer invoice recognition rather than text recognition by creating a context where payment information is specifically evaluated.

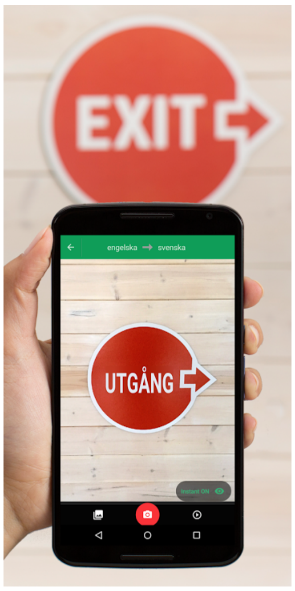


Figure 1: A translation in Google's mobile application, "Google translate".

5 Problems to be solved

Two main issues arise when designing a solution. How can an application read analogue data, and when it does, how will it know what the data represents?

Converting the analogue invoice. An existing API will allow us to perform the actual character recognition, leaving us mainly with the process of enhancing the image. Pre-processing of the image will focus on unifying the colors in the image in order to create an unmistakable difference between the text and the background. We will also need to divide the image into smaller blocks as the fields are never organized in machine written lines. Once the image is divided into small readable blocks all that's left to do is read the data, which as stated will be extracted by using an existing API.

Interpreting the data. Once the OCR is complete we will have a set of strings representing the text on the image. Sometimes these strings will be slightly misread due to similarities between different characters, poor image quality or dirt on the scanner used to generate the image. This will need to be corrected somehow, we will attempt to use previously read invoices to fill in the blanks wherever possible, followed by a prompt to have the user confirm the correction. Should this also fail the user will then have to restart the process or simply fill out any missing fields manually. The data also needs to be identified and paired with the correct field, i.e. the price needs to be the same as the price on the image. This will be accomplished by a combination of keywords and pattern recognition of the formats.

6 Method

The application is developed using C# in the Visual Studio environment. We chose C# as we thought that the application would be easier to expand with new features if it is written in a object oriented fashion on an actively developed platform. We considered development in Java as it is easier to distribute across multiple platforms, however it does require the JRE, java run-time environment, to be installed on the targeted system.

The application will only be able to run on Windows-systems, which should be sufficient as it is intended for smaller companies who tend to use Windows. This according to measurement done in April 2016 by the web traffic analysis tool StatCounter [13]. Considering the choice to focus on a Windows desktop application, Visual Studio is an appropriate development environment since it is also produced by Microsoft, for Windows. Visual Studio offers tools to create elements that reflect the operating systems graphical flavor, such as Windows Forms, which will increase the familiarity factor.

Pre-processing, OCR and pattern recognition. Pre- and post-processing is performed on the image in order to help the OCR-process. It is mainly done by changing the colors to create a clear contrast between text and background. The image is also cropped down to several smaller images containing only a few words to speed up the OCR. Once the image has been enhanced, the blocks are read one by one and transformed into digital characters. Once the conversion is complete, manipulation and identification of the strings is needed. The digital strings generated by the OCR are matched to regular expressions in order to find a matching type of information, e.g. the banking giro. Should a given string be unmatched by every regular expression that string is considered trash and is not used. Once all the data from OCR have been used, a data bank of previous translations is used to fill in any blanks. If an empty field is found the corresponding data from an old invoice will be used, provided the sending company in question exists in the data bank. This helps prevent empty fields resulting from sub-optimal translation. Once the application is done it allows the user to modify or correct any fields necessary.

6.1 Choosing an API

Existing OCR solutions are quite powerful and offer support for different kinds of functionality. During the project we implemented versions using different engines as a part of our development process. The Tesseract OCR engine was found out to be very flexible, and with an ability for self-learning. The other problem is to correctly prepare the image for text recognition and capture coordinates of each text-line. In this case we discovered the OpenCV library, which contains possibilities for working with image manipulation and Tesseract OCR is included in the OpenCV library.

Tesseract API. Tesseract is a *FOSS, free and open source software,* solution for OCR. It was originally developed by Hewlett-Packard (HP) and later overseen by Google. It was first developed for English but over the last few years it has been developed to support many more languages. In a report written by Nick White it is stated that Tesseract is a flexible OCR software solution that has a wide code base and a large, engaged community [14].

OneNote API. Microsoft wanted their customers to easily create and share their ideas using Tablet-PCs, so they created an API for OCR [15] within Microsoft OneNote, to accomplish this. The API showed good results, even with bad quality images, but there was one problem that canceled out these benefits. In order to use the OneNote API, the targeted client machine needs to have an installation of the OneNote application. Furthermore the OneNote application is launched every time the API is used, and OneNote will then display the image before the the API can begin performing OCR on said image. We considered it an nonviable option to force every target client to have access to a third party application. Not to mention that having to view each image in a third-party application would disrupt the usability of your own application.

Google Cloud Vision API. Google offers a powerful API for image and object recognition, which is able to recognize objects in photos and is even able to determine emotions based on facial expressions in humans [16]. While the power of the Vision API is impressive it has a major drawback rendering it an unsuitable option for our application. The API handles a lot of processing server-side forcing any prospective programmers to send the images to Google's server. This might cause security issues or at the very least confidentiality issues as internal payment and bookkeeping information would have to be regularly shared with a third-party. Furthermore this processing by Google's server is not free, they charge a fixed amount per threshold depending on how many images you send for recognition, the nature of the recognition also weighs in, although OCR is one of the cheaper ones. Forcing a potential customer to rent processing power from a third-party seemed like a poor choice if the application potentially is to be distributed.

OpenCV. The OpenCV API [17] is designed for image-processing and is able to identify objects and characters in both pictures and video clips. It also provides tools to enhance the image before reading it allowing you to crop the image and unify the colors to generate a clear contrast. Furthermore it is available on multiple platforms and in different programming languages, such as Java, Python and C#. The flexibility and image-enhancing tools are what set this one aside from the others.

7 System structure

The flow of the application can be broken down into several small steps. It begins when the user scans an invoice and ends when the user accepts the extracted data. In between, the application will pre-process, read, match the strings and then fill in the blanks from a data bank, called company dictionary, of previous invoices while simultaneously updating the data bank.

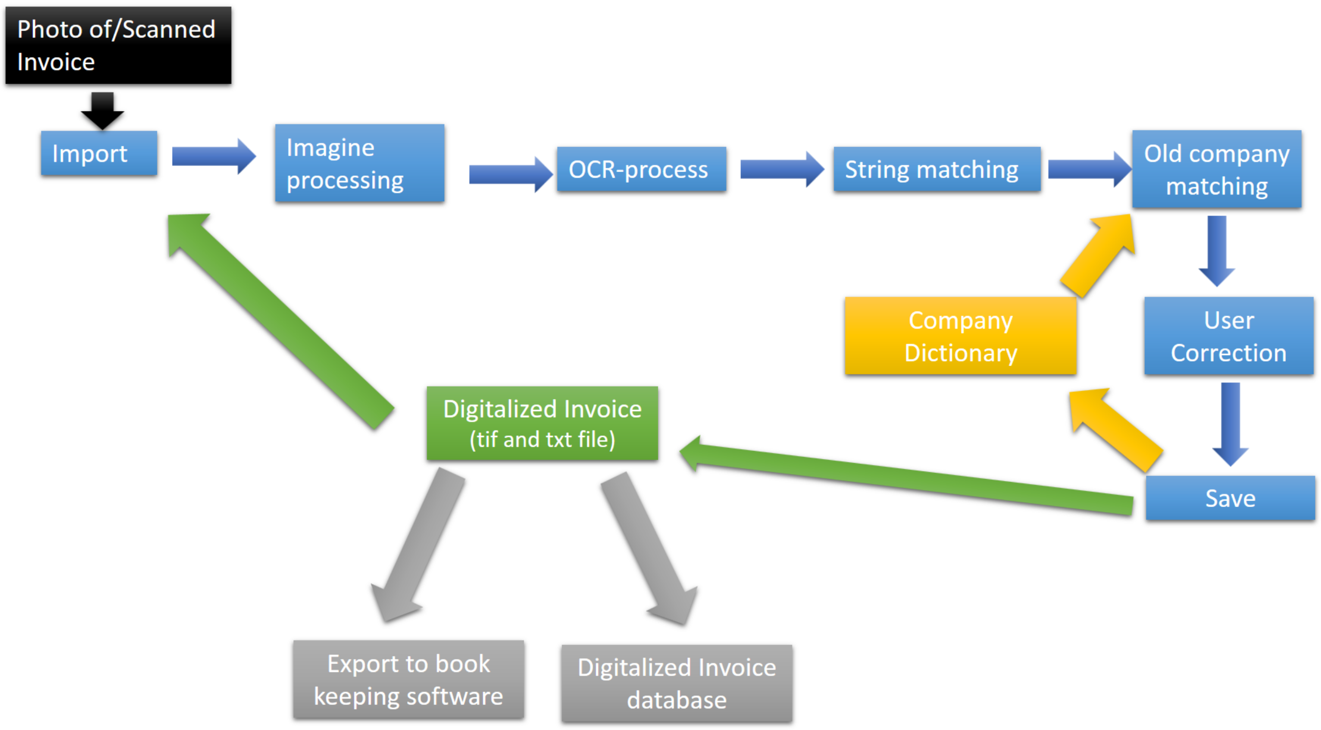


Figure 2: A flowchart displaying the applications stages. The black box represents the first stage in any convertion by the application.

Photo of/scanned pictures. Upon receiving a paper invoice, the company must scan the invoice in order to make use of the application. The scanned image must have a resolution of at least 300 *DPI, Dots Per Inch,* for optimal results and the file format must be *TIF, Tagged Image File*.

Import. After launching the application the user can import a TIF-file using the original Windows file browser. The imported image is loaded to a bitmap. The picture will be shown to the left in the window. An already digitized invoice should also be able to be imported. In that case a txt-file will be selected and loaded.

Image processing. The bitmap from the imported TIF-file is split into several bitmaps. Each bitmap might have a small piece of the text on the invoice.

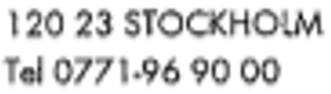

120 23 STOCKHOLM
Tel 0771-96 90 00

Figure 3: An example of a bitmap containing a postal address and phone-number

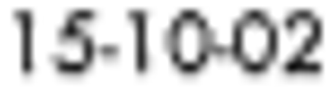

15-10-02

Figure 4: An example of a bitmap containing a date

OCR-process. Once pre-processing has finished, the application attempts to read the text in it, using the Tesseract OCR. Corresponding digital strings are generated and passed to pattern matching.

Pattern matching. Once the digital strings have been generated they are matched to a set of patterns to determine what type of data has been translated. e.g. if a string consisting of three sets of digits divided by dashes is found then it is most likely a date, i.e. NN-NN-NN.

Old company matching. If for some reason a given field was not converted or identified correctly, the application searches for previously digitized invoices from the same company. If a match is found the missing field is filled in with the old data. This step only works with fields that describes static information such as the service-providers address or banking giro.

User Correction. Once the application has performed the data extraction, the user needs to accept or alter the resulting data. If the user detects an incorrect field or if one is blank, the user can simply correct it.

Save. Once the data has been approved it is saved in whichever folder the user chooses. Any new data about the company who sent the invoice is stored in the company dictionary to enhance company matching in future extractions.

Digitized invoice. The output of the application is called the ”digitized invoice”. It is a folder containing the image of the original invoice and a corresponding txt-file with the extracted data. This extracted data is organized with the name of the image file on top, the rest are pairs of lines, the first of which is an identifier and the second the data being referred to by the first line.

Company Dictionary. It is likely that the company using the application will get invoices from the same service providers on a regular basis. To improve performance, a data-file containing company specific information from previous invoices is used, this way, the information will not have to be repeatedly recognized. Company address information and company financial information are kept here. This helps the application to provide more efficient processing for the next invoice from a particular company.

This application is more of a new combination of existing API's and libraries. It will fill the gaps between bookkeeping software, internet banking applications and paper invoices. Each application user has the same status, there are no administrative rights.

8 Requirements and methods of evaluation

8.1 Basic functionality

The final product should be able to take an image file containing an invoice as input. Acceptable analogue input should be digitized and stored. The user should be able to change any given field manually prior to saving. A digitized invoice should be exportable to be compatible with common bookkeeping systems.

8.2 Extraction error

Reading and interpreting text with OCR is not flawless, and thus a quota for acceptable error occurrence was needed. It was decided that a 95% success rate was acceptable. Inside information from discussions with the client company described that ”a manual correction every 20th invoice is quite manageable”. An extraction is considered successful if every relevant field for the given invoice is extracted correctly. It is successful even if an unimportant field is not extracted correctly, e.g. if the company has no interest in paying a given invoice via postal giro, digitizing the postal giro is not important. For evaluation purposes a fixed set of critical fields are required for an extraction to be considered successful.

- Bookkeeping details
  - Invoice identification number
  - Reference
  - Pay day
  - Invoice created
  - Currency
  - Price before tax
  - Rounded
  - Total amount
- Financial information
  - IBAN
  - VAT

  - Payment refrence
  - Invoice number
  - BankGiro
  - PlusGiro

- Contact information
  - Company name
  - Organization number
  - Phone
  - Webb
  - Address
  - Postal index
  - City
  - Country

8.3 Input variations

The greatest challenge for the application is the number of different possible invoice layouts. The user should as infrequently as possible need to correct the extracted information. To create a reasonable restriction in the possible invoice variations as input, we have put a limit on the evaluation to a set of invoices that we originally obtained from Crowderia.

9 Designing and implementing the application

Image enhancement To get lines that can indicate text areas, the image goes through multiple filters that are provided by an external image processing library, OpenCV. In order to remove light-marker noise that interrupt text lines, image-enhancing operations are performed on the image. Erode and Dilate are two basic morphological filter-operations that process objects in the input image based on characteristics of its shape. These two filters are part of one single process where Dilate enlarges characters and Erode shrinks them to original size, thus filling in missing pixels. This enhancement process with Dilate and Erode will be applied after every other filter to counteract the destruction that they cause.

At first, a binary filter is applied on the image to turn all foreground objects to 0 (black) and the background to 255 (white). To achieve correct and fast contour recognition the image is scaled down and Blurred. The image will then be scaled back up. A Closing filter is applied to separate blocks, usually this way a large text-block may be split into smaller parts. Lastly Binary Inverse is applied one more time to recreate the original background/foreground colors.

Image cropping Each text-block in the picture is captured within a rectangle shaped block. All captured blocks are checked to be below the maximum allowed size, blocks that overlap are merged since multiple different blocks can occur within the same area. The goal is to separate each line of text, but this is not always possible due to small margins between lines in different parts of the image. Merging text-blocks vertically is possible, but smaller text is harder to recognize because of the lower precision you get from low resolution and multiple text-lines.

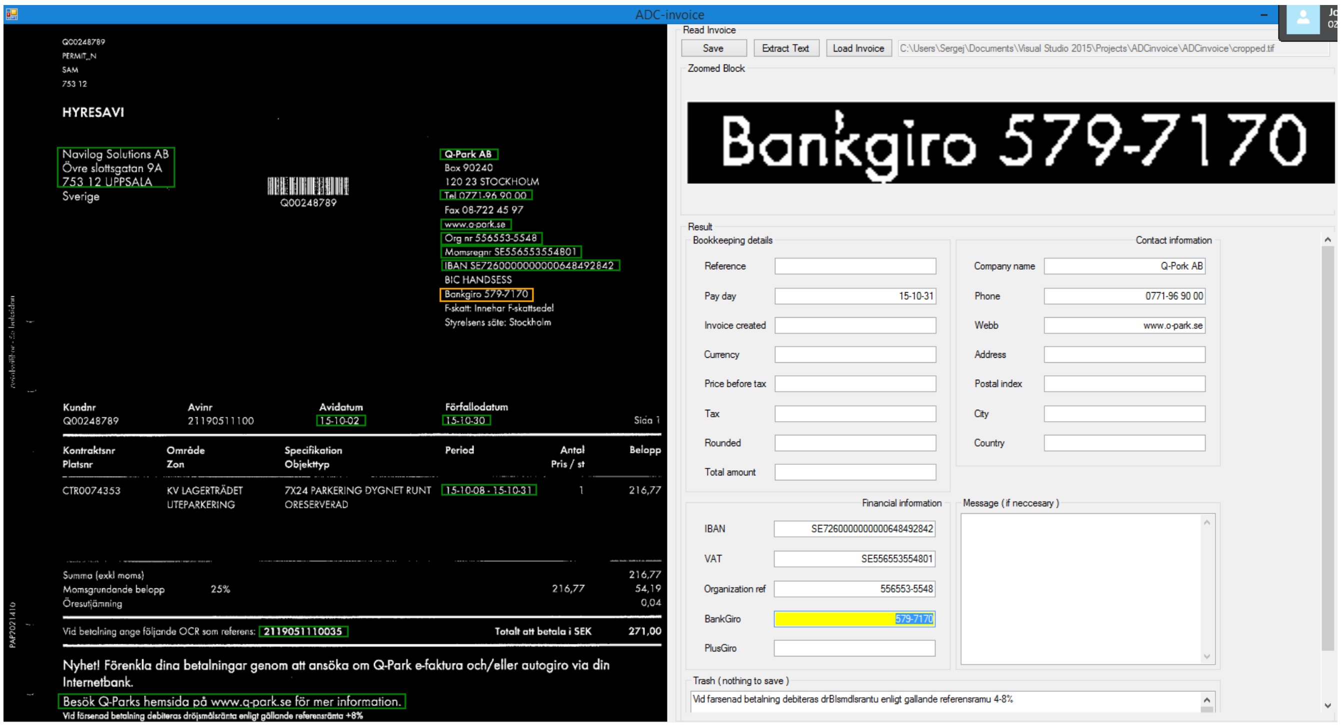
Pattern matching. The strings returned by the OCR are matched against regular expressions [18] containing the patterns relevant for bookkeeping. Strings that match a regular expression are extracted and tagged with the corresponding invoice label. Every pattern is checked in a predefined order for every block of text. The order follows the longest pattern first, to prevent possible matches where a pattern is also a sub-pattern to a longer one, causing mismatches. As an example, IBAN numbers typically have twenty-four characters out of which the first two are letters and the last twenty-two are digits. VAT numbers typically have 14 characters where the first two are also letters while the last twelve characters are numbers. As both patterns differ only in the length of the numbers, a pattern that captures any string that begins with two letters followed by digits would not be able to separate the labels. Because of this it is better to match with the length of the tailing digits as well. Any pattern that is too short can then be eliminated, leaving only the IBAN number as a possibility. A full list of patterns are attached in appendices A and B.

10 How to use ADC Invoice

The user needs to do three things when using ADC Invoice. An invoice needs to be scanned, imported to the application and the results must be inspected and accepted.

Step 1: Scan the invoice to a TIF-file. It all starts with a company getting a paper invoice, this invoice might be scanned or someone might take a photo of it. We recommend that it is scanned in at least 300 DPI in order to make the OCR-process more efficient and the file format should be TIF.

Step 2: User inspection. Following every extraction a user must validate the results as a safety measure against misinterpretations by the application. An image of the invoice is available as aid in inspecting the results. The user must inspect the information generated in the fields to the right and decide if something is wrong or missing. Should an error occur, the user must manually type in the correct information before validating.


ADC-invoice

Q00248789
PERMIT_N
SAM
753 12

HYRESAVI

Navilog Solutions AB
Övre slottsgatan 9A
753 12 UPPSALA
Sverige

Q00248789

Q-Park AB
Box 90240
120 23 STOCKHOLM
Tel 0771-96 90 00
Fax 08-722 45 97
www.q-park.se
Org nr 556553-5548
Momsregnr SE556553554801
IBAN SE7260000000000648492842
BIC HANDSESS
Bankgiro 579-7170
F-skatt: Innehar F-skattsedel
Styrelsens säte: Stockholm

Kundnr	Avinr	Avidatum	Förfallodatum	
Q00248789	21190511100	15-10-02	15-10-30	Sida 1

Kontraktsnr Platsnr	Område Zon	Specifikation Objekttyp	Period	Antal Pris / st	Belopp
CTR0074353	KV LAGERTRÄDET UTEPARKERING	7X24 PARKERING DYGNET RUNT ORESERVERAD	15-10-08 - 15-10-31	1	216,77

Summa (exkl moms)			216,77
Momsgrundande belopp	25%	216,77	54,19
Öresutjämning			0,04

Vid betalning ange följande OCR som referens: 2119051110035 Totalt att betala i SEK 271,00

Nyhet! Förenkla dina betalningar genom att ansöka om Q-Park e-faktura och/eller autogiro via din Internetbank.
Besök Q-Parks hemsida på www.q-park.se för mer information.
Vid försenad betalning debiteras dröjsmålsränta enligt gällande referensränta +8%

PAP2021410

Read Invoice
Save | Extract Text | Load Invoice | C:\Users\Sergej\Documents\Visual Studio 2015\Projects\ADCinvoice\ADCinvoice\cropped.tif

Zoomed Block

Bankgiro 579-7170

Result

Bookkeeping details

Reference	
Pay day	15-10-31
Invoice created	
Currency	
Price before tax	
Tax	
Rounded	
Total amount	

Contact information

Company name	Q-Pork AB
Phone	0771-96 90 00
Webb	www.o-park.se
Address	
Postal index	
City	
Country	

Financial information

IBAN	SE7260000000000648492842
VAT	SE556553554801
Organization ref	556553-5548
BankGiro	579-7170
PlusGiro	

Message (if neccesary)

Trash (nothing to save)

Vid farsenad betalning debiteras drBlsmdlsrantu enligt gallande referensramu 4-8%

Figure 5: ADC Invoice handling an invoice from Q-park is. The extraction is complete and is ready for correction by the user.

Step 3: Save and export. Once the extraction is complete and validated the user must decide where to send the results, they can be saved internally as a txt-file next to the image or exported to a template which can be sent to a bookkeeping application.

11 Results

The product application supports importing of images, extraction and interpretation of the invoice information. The user can inspect and correct the extracted information. At the end of the process the information is saved together with the original image of the invoice.

11.1 Time consumption.

Three people, two of them familiar with bookkeeping, were given an invoice and instructed to type a predetermined set of fields onto a text editor. The timed results measured; 4 minutes and 5 seconds, 2 minutes and 41 seconds and 7 minutes and 43 seconds. An average of 4 minutes and 50 seconds. ADC Invoice performed the same work in 35 seconds. However none of these results includes correctness of the data. If all data from an invoice image is to be extracted, ADC Invoice performs the task on average 4 minutes and 15 seconds faster.

11.2 Accuracy

Correctly extracted and interpreted fields are: payment reference, postal and banking giro, dates, organization number, postal number and all price figures. This result correlates to 13 of 25 fields in the list under 8.2. The total percentage, 52%, is unfortunately lower than the 95% target. However, bookkeeping details that change for each invoice but share company information with other invoices, can be used to auto complete previously known details. The application then reaches an accuracy around the required 95%.

11.3 User experience

A cognitive walkthrough gave a positive response, the user subject felt that the GUI was very intuitive and comprehensible. During the evaluation we received some feedback regarding the potential of future works such as management of a company dictionary that could simplify interaction. Graphically, the information feedback that the user received was satisfactory considering the detailed information of the actual OCR process and animations to indicate the process has not stalled.

12 Evaluation

Because the application is intended to reduce the workload of the user, the evaluation must take both technical and human interaction aspects into consideration. To measure how well the application performs on a technical level, we have inspected how many invoices were successfully extracted. Any fields that contained non-ASCII characters such as the Swedish letters å, ä, ö were omitted from the result as these characters were expected to be misinterpreted. The human interaction aspect has been evaluated using a cognitive walkthrough of the application.

12.1 Features

Input types. The application accepts input in the form of a TIF-file, which can contain different image formats. However if the image offered as input is not marked as TIF, and is instead an incompatible image file, the application will not accept it. Thus the input requirement was partially met given that most image files can be used as input as long as they use to the specified extension.

Digitize the input. Assuming the input format is acceptable, any text contained within it is processed, however there are remaining issues with pattern matching that caused some of the digitized data to be discarded as trash. Since the application is able to find and read any text given acceptable inputs, the requirement is met.

Validation of extracted data. Since successful correction is not guaranteed in every case the user is allowed to change any field the application fills with extracted data. Once the application has extracted everything it finds in the input, the application pauses. Before the results are stored the user has a chance to validate them. Thus this requirement is fully met.

Exporting extracted data to bookkeeping. One required feature for the application was the ability to export digitized invoice to a bookkeeping application. This requirement was de-prioritized during the development due to lack of API specifications for incorporation with the discussed systems and was never implemented, leaving requirement that digitized invoices can be exported unmet.

12.2 Success Rate

To evaluate if we met the targeted success rate (of the information mentioned in the list under section 8.2), an extraction was attempted for each of the predetermined invoices. If every critical field was correctly extracted that invoice was considered a success. This inspection was performed manually with the image of the invoice as reference. The success rate turned out to be 52%, which was considered a failure as it was below the 95% target.

12.3 Human interaction

The human validation step we have developed is adapted to a user that is already familiar with the invoice format and can inspect the content without instructions from the program. However the analogue information will be available for comparison within the interface to aid inspection. These circumstances allow an end user to complete invoice extraction.

Since a lot of the information is context-sensitive, the persona that is most relevant to our evaluation does regular handling of invoices. In order to evaluate the user experience of the product we performed a cognitive walkthrough with an individual familiar with invoice bookkeeping. We stepped through the process of performing a complete digital conversion of an invoice, we were able to perform the task according to the stakeholders expectations.

13 Discussion

Character support. The Tesseract OCR engine was able to adequately extract all of the characters we expected, and present them in a computerized string. The resulting strings were not flawless though, since a few incidents of the letter 'o' being interpreted as the number 0 were recorded. This error only appeared to occur when letters and digits were mixed in a string. An interesting comparison is the application offered by Swedbank, which reads payment dates, reference and price. This application is designed to have the user point out the correct number before it is scanned, thus it can expect the input to be only numbers, effectively preventing the issues that can appear when digits and letters are matched. A similar realization was that non-ASCII characters such as the Swedish letters å, ä, ö reduced the overall quality of the OCR-process. Thus we decided to drop support for non-ASCII characters.

Layout problems. The toughest challenge of the project has been to manage the lack of common structure in invoices. The same information can be located on different parts of an invoice, and the structure of the information can be given in different formats. A date can for example be written as 16-03-21 or 21 of March 2016. A human may realize instantly that the underlying information is the same. It is a lot harder to design an algorithm that can recognize these variations. The approach taken is to go through all recognizable numbers and try to match these numbers against known patterns to identify their purpose. A downside to predefined patterns is that a following invoice could be written in a slightly different way, and suddenly the result of the extracted information might be of poor quality due to failed pattern recognition. To compensate for the difference with this new invoice, we would have to rewrite the code to adapt to new patterns as well. This can be done repeatedly, and the risk of getting an invoice that is not perfectly supported by ADC Invoice would decrease, but we will never know exactly what the next invoice will look like or if it is supported by ADC Invoice.

Accuracy of the results. If all fields are taken into account, only 52% is recognizable. When comparing results of the fields that are unique for each invoice, the required 95% is reachable. It should be taken into consideration that the invoices used for this evaluation are randomly picked Swedish invoices. If this evaluation is redone with another set of invoices, it is highly possible that an inferior result will be produced. This is due to the problems with different layouts.

Time consumption of the results. When comparing the time in 11.1 for a human typing in the information with the ADC Invoice extraction, the human was outperformed. One thing to keep in mind is the time it takes to go through the scan process and if necessary, convert it to a TIF-file. Therefore one could argue that the comparison is unfair. In the case of incorporating our software to a business, it will have to be considered how to make the scanning and importing steps efficient.

14 Conclusions

This project undertakes the challenge to make the bookkeeping process more adapted to digital systems. Our efforts have been dedicated to the development of an application called ADC Invoice that will translate analogue paper information to digital form. The performance of the final product presents clear advantages to manual typing, the measured results were almost five times as fast. Although the recognition process of the application is not completely accurate, ways to supplement the optical character recognition by auto completion is used. The supplementing algorithms can significantly reduce the amount of information that needs to be extracted.

Some of the challenges with invoice layout are inherently hard to solve due to the large amount of variation in the information present as input. From a cynical viewpoint, we are unable to give any guarantees of what will be correctly interpreted, since a good model of a generic invoice is hard to find. The ideal circumstance of an invoice digitization with ADC Invoice is one where the input layout has previously been empirically confirmed to be recognizable.

There are currently many OCR applications available that attempt general recognition of texts. We have been able to introduce a solution that is specialized on invoices. The advantage our solution offers is the capability to run the OCR software locally, without specialized hardware, which gives you flexibility without compromising integrity by sending sensitive information to external servers.

Our application is fully capable of recognizing common invoice patterns assuming the characters are correctly interpreted and given a repeated layout format our solution will significantly reduce the time consumed by the digitization process. Although our implementation was developed with a business in mind, the final product is not only targeted towards small companies, but could be used to manage private invoices completely unrelated to our client since the interface is adapted to a generic user.

15 Future work

Future work pertaining to our results can be split up into two groups. An improvement on already existing features, such as accuracy or a faster algorithm, or adding new features extending the range of available actions and functionality to communicate with a wider selection of software.

15.1 Improvements

The ADC Invoice string matching step, [fig. 2 in 5], is currently not optimal. The pattern matches are based on patterns we could derive from the limited number of invoices that were available but we are aware that there are invoices that use different patterns for some fields. The main issue with the invoices is that the layouts can differ greatly and thus common denominators are scarce. Covering as many patterns as possible will require multiple patterns for each given field without having patterns from different fields being so similar that they could be mismatched. The invoices that were made available to us were exclusively of Swedish origin which begs the question of how ADC Invoice would handle an invoice from another country. Offering reliable support for any invoices received should be a prioritized improvement.

A possible workaround to the issue of finding patterns could be to attempted using a data bank of commonly used terms to act as keywords for the fields on any given invoices. This could remove some of the difficulties in finding patterns to match specific numbers, such as the IBAN number, but would also require every field to be grouped with its corresponding keyword when the application splits up the text into smaller blocks.

15.2 Extensions

The two ways to export the data [fig. 2 in 5] extracted from an invoice are currently not implemented. These features are important since the objective is to make as much as possible of the invoice-processing automatic. It would be a lot faster to simply confirm the result and have it automatically sent to bookkeeping, rather than doing it manually.

Export to bookkeeping. Storing payment history in your bookkeeping system is of high priority in most businesses. Thus an extension of ADC Invoice that allows the user to perform this task more quickly is very useful, in fact we originally planned to offer such a feature, however it was not finished in time. Developing this feature will require knowledge of how data is inserted into the given bookkeeping application, we specified Visma and Fortnox's solutions but the concept is the same for any other bookkeeping software. If such knowledge cannot be obtained, for example if the creators of the bookkeeping software do not offer a public API, then a convention for passing the data will be needed. It could be something as simple as a string with the data organized in a specific order or something more complex like a heavily encrypted data file to prevent fake insertions from third parties.

Save to database. Generating a database of previously processed invoices could make it easier to search invoice history data of the business. Currently any invoice that is read by the system is saved with a file containing the extracted data, without any information considered trash. In addition a database could be constructed, allowing for some basic searches for traits that are context specific, e.g. date of invoice, price paid or information about the service provider. This could potentially turn into an advanced feature allowing multiple filters and perhaps a way of determining if a new invoice is an exact replica of an already processed and available duplicate, effectively preventing multiple entries.

Batch processing. We noticed when scanning multiple papers in a batch that commonly the digital output was a PDF-file with each image on a separate page. Thus it would be useful if ADC Invoice could take such a PDF as input rather than a single TIF-file as it does currently. The application could then separate each image from the PDF into a set of corresponding TIF-files. This would enable batch processing of invoices. Since an invoice can be read and extracted in an average time of 30-40 seconds, it would be beneficial time wise to have the application read and extract 10 invoices at a time. Once the process is finished they can confirm and, if needed, correct the invoices one by one without having to wait for the next invoice to be read.

Accounting Accounting means to decide to what account an invoice should be charged. If the invoice is a monthly payment for a mobile subscription the invoice should be charged to the account for phone expenses. Discounting is like accounting when the expenses target more than one account. We decided to not implement features to support discounting because it is not strongly related to our project task, but rather a potential expansion and future work. To associate invoices, or part of invoices, to an account the program has to interpret the context of the text. For example to recognize that a stapler is part of the category Office Supplies, i.e. if your company has a service provider X that your company only buys office supplies from, then it is possible for the application to associate the company name X to office supplies and that way get the right account. However if you do not buy only office supplies from company X then an association requires knowledge about what the product or service is.

References

[1] Swedbank, “Youtube: The mobile bank in 5 minutes,” November 2014, accessed Apr 2016. [Online]. Available: https://www.youtube.com/watch?v=CUdvyz2z92Y

[2] Visma, “Bye bye manuell attestering,” pp. 1–4, Jul, accessed from Visma’s homepage in May 2016.

[3] Fakturaportalen, “Om e-fakturor,” accessed June 2016. [Online]. Available: http://www.fakturaportalen.se/sv/e-invoices

[4] S. Mori, C. Y. Suen, and K. Yamamoto, “Historical review of ocr research and development,” *Proceedings of the IEEE*, vol. 80, no. 7, pp. 1029–1058, Jul 1992, doi:10.1109/5.156468, ISSN:0018-9219.

[5] Crowderia, “About crowderia,” accessed Apr 2016. [Online]. Available: http://crowderia.com/about/our-story/

[6] Lexmark, “About lexmark,” accessed May 2016. [Online]. Available: http://www.lexmark.com/en_us/about.html

[7] ——, “Invoice processing for oracle,” accessed May 2016. [Online]. Available: http://www.lexmark.com/en_us/solutions/financial-process-automation/solutions/erp-integration/automation-for-oracle-e-business-suite/invoice-processing.html

[8] Oracle, “E-business suite store page,” accessed May 2016. [Online]. Available: http://www.oracle.com/us/products/applications/ebusiness/overview/index.html

[9] Fortnox, “Fortnox fakturatolkning,” accessed May 2016. [Online]. Available: https://www.fortnox.se/program/plustjanster/fakturatolkning/

[10] MAKEUSEOF, “Top 5 free ocr software tools to convert images into text,” accessed Apr 2016. [Online]. Available: http://www.makeuseof.com/tag/top-5-free-ocr-software-tools-to-convert-your-images-into-text-nb/

[11] Voice4u, “Vocie4u tts - a text-to-speech app with natural sounding voices,” accessed Apr 2016. [Online]. Available: voice4uaac.com/tts

[12] G. Play, “Google Översätt,” October 2015, accessed Apr 2016. [Online]. Available: https://play.google.com/store/apps/details?id=com.google.android.apps.translate

[13] StatCounter, “Statcounter global stats - top 7 desktop oss on apr 2016,” accessed May 2016. [Online]. Available: http://gs.statcounter.com/#desktop-os-ww-monthly-201604-201604-bar

[14] N. White, “Training tesseract for ancient greek ocr,” *eutypon 28-29*, pp. 1–5, Feb 2013.

[15] D. Dorothy, “Tabletnotes,” in *Proceedings of the 46th Annual Southeast Regional Conference on XX*, ser. ACM-SE 46. New York, NY, USA: ACM, 2008, pp. 386–389. [Online]. Available: http://doi.acm.org.ezproxy.its.uu.se/10.1145/1593105.1593207

[16] G. C. Platform, “Google cloud vision api - quickstart,” April 2016. [Online]. Available: https://cloud.google.com/vision/docs/getting-started

[17] S. Brahmbhatt, *Practical OpenCV*, 1st ed. Berkely, CA, USA: Apress, 2013.

[18] donetperls.com, “Ragex,” accessed May 2016. [Online]. Available: http://www.dotnetperls.com/regex

A Patterns for string matching

```
string[] patterns = {
        @"(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})
                ([/[a-zA-Z]\.-]*)*\/?",
        @"[a-zA-Z]\s*[a-zA-Z](\s*\d){22}",
        @"[a-zA-Z]\s*[a-zA-Z](\s*\d){12}",

        @"(\(0\)|0)((\s*\-)?\s*[0-9]){9}",
        @"[0-9]{2,4}(\-[0-9]{2}){2}",
        @"[0-9]{6}\-?[0-9]{4}",

        @"([0-9]\s*){4}((\.(\s*[0-9]){3}\s*\.(\s*[0-9])
                {4}\s*\-(\s*[0-9]){2})|(\-(\s*[0-9]){6,8}))",
        @"[1-9](\s*[0-9]){5}\s*\-(\s*[0-9]){7}\s*\-[0-9]",
        @"([0-9]\s*){3,4}\-(\s*[0-9]){4}",
        @"([0-9]\s*){6}\-\s*[0-9]",

        @"([A-Z][a-z0-9]*\s*[\-\&]*\s*)+\sAB",
        @"([0-9]\s*){5}[A-Za-z]+",
        @"[A-Z]([a-zA-Z]\s*)+[0-9]{1,5}(\s*[a-zA-Z])?(?![\-0-9])",

        @"(SEK|LKR|\$)",
        @"([0-9]\s*)+[\,\.](\s*[0-9]){2}",
        @"(\s*[0-9])+"
};
```

B Pattern types

```
public enum Pattern {
        WEBSITE,
        IBAN,
        TAX_REGISTRATION_NUMBER,

        PHONE_NUMBER,
        DATE,
        ORGANIZATION_REFERENCE,

        OBJECT_REFERENCE,
        INSURANCE_NUMBER,
        BANKGIRO,
        PLUSGIRO,

        COMPANY_NAME,
        COMPANY_INDEX_AND_CITY,
        COMPANY_ADDRESS,

        CURRENCY,
        PRICE,
        OCR_REFERENCE,

        BACKUP_NUMBERS
};
```