This blog is a comprehensive guide on exporting information from scanned PDFs to Excel

Information Explosion and the Use of PDFs

Information is everywhere. According to statistics, more than 1.7MB of data is created every second in the year 2020. If this trend continues, we would be having 463 exabytes of data by the end of 2025. This data can be anything, say, the information collected by self-driving cars, company-related documents, emails, photos etc. Out of these, to store text-related data, PDFs are most frequently used. Some common examples of PDFs include books, invoices, tax forms, logistic information, and many more.

But there's one problem here! Many people find it confusing to parse or extract important information from PDF documents; therefore they find ways to migrate data from documents to Tabular data (mostly Excel sheets) to utilise information and bring out meaningful insights. In this column, we'll learn how we export PDF information to Excel sheets through different techniques. We'll also look at how OCR and Deep

Learning can help us automate the entire process of extracting information from PDFs.

Before getting started, here's a quick outline of the post:

The Problem with Converting PDFs to Excel

PDFs are usually one of the most readable formats for viewing data. But converting them to Excel sheets is a hard task to accomplish because:

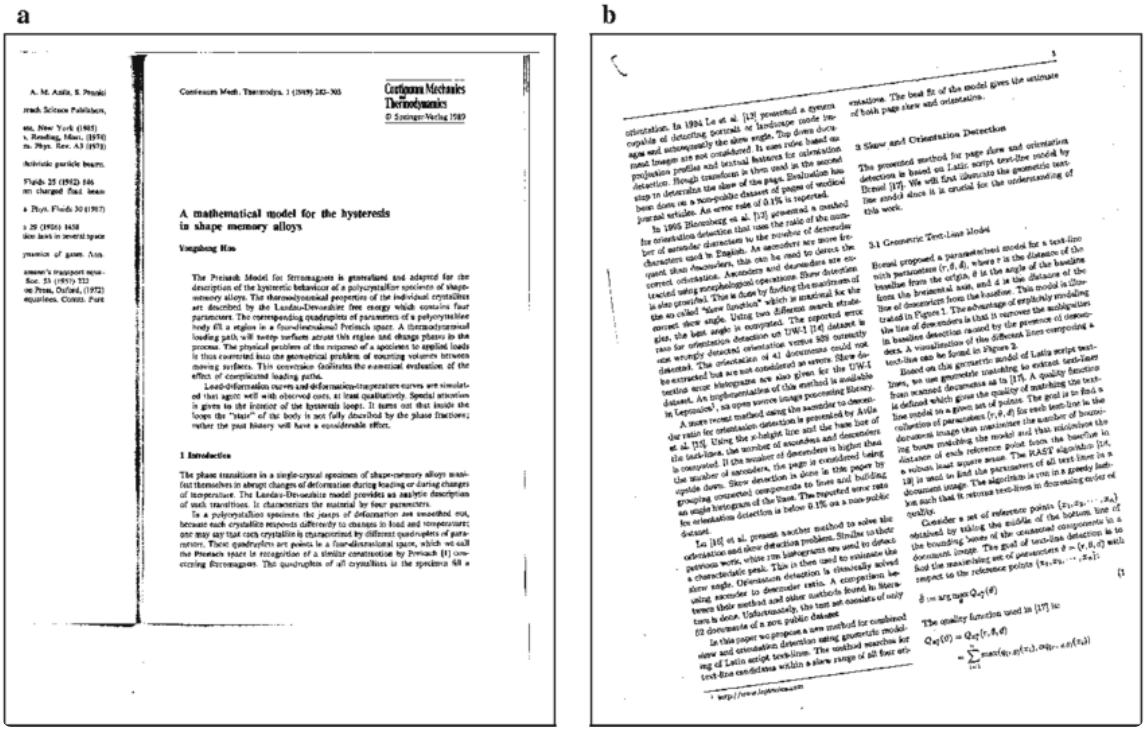
- We need a format with simple primitives and no structured information
- There's no equivalent of a table component in PDF files as tables are created with straight lines and coloured backgrounds
- As tables in PDFs are drawn like images, detecting or extracting a table is a complex process: - We understand templates in terms of shapes, the position of text, relationship between lines and text, etc,.
- PDFs created by digital image or by scanning a printed file have distorted lines and no textual elements

The whole endeavour seems desperate, but as we are going to see, sometimes we can actually extract information from these PDF files, too.

By looking at the above problems, there are two types of PDFs containing tables:

1. Tables with textual data (generated electronically)

2. Tables with scanned images (generated non-electronically)



Low resolution/ disoriented scanned PDFs (non-electronically)

Looking to export information from scanned PDFs to Excel sheets? Head over to Nanonets to automate the process of exporting from PDFs to Excel...

Get Started

How does Exporting Scanned PDF to Excel Work?

PDF files are purposed solely for viewing data and not for manipulating it. Therefore, exporting PDF data to Excel sheets is one of the most tiresome and complicated tasks.

Most users or developers start by browsing through some of the online tools out there to perform this task. But they aren't accurate or capable enough to parse through complex PDF formats. Also, these tools are not free to use, and they are limited to daily or monthly usage.

To understand the working of moving PDFs to Excel, we'll have to first verify if the PDFs are made electronically or not. When the PDFs are electronically generated exporting PDF to Excel is quite straightforward. It involves exporting the data into a Word document and then copying it to an Excel workbook. In the second case, when the PDFs are not generated electronically (say if they are captured through phones or downloaded from Email), the process is quite tricky. Below is a detailed walkthrough of how the exporting process works:

- Firstly, PDF to Word/Excel/Direct Text converters are used to copy the information we need. In this case, the result is often messy if the PDFs follow any templates or if there are any tables.
- OCR (Optical Character Recognition) engine is used to read the PDF and then to copy its contents in a different format, usually simple text. Quality varies between the OCR engines and often the licences are not free. You could always go with the free and open-source Tesseract OCR but it requires some programming know-how. Or you could try OCR tools that run on zonal OCR or more advanced AI/ML based OCR algorithms.
- Some additional programming is required to process the text into the required format or store them in tabular format. If you're a developer and familiar with coding, PDFMiner (Python-based) or TIka (Java-based) can be used.
- Lastly, we'll have to write code snippets to push formatted data to Excel or configure online APIs if we're using Google Sheets.

Methods to Detect Tables in Textual PDFs

Now, let's start by discussing methods for extracting tables from PDFs when they are electronically made. To accomplish this task, we have two techniques: Stream and Lattice. These techniques were first revealed and improved by tools like Camelot and Tabula. We'll also go through some of the examples using these tools and see them in action in the further sections.

Detecting Tables Using Stream:

This technique is used to parse tables that have whitespaces between cells to simulate a table structure. Basically, identifying the place where the text isn't present. It's built on top of PDFMiner's functionality of grouping characters on a page into words and sentences using margins.

Below is a quick explanation of how this technique works:

1. Firstly, the rows are detected by making rough guesses based on the y-axis position (i.e., height) of some text. Basically, all text on the same line is considered to be part of the same row. To read more about this, you can go through Anssi Nurminen's master's thesis on finding table locations in PDFs.
2. Next, the text is grouped in columns based on some heuristics. In a PDF, each word is in its own position, so basically, words are put in the same group if they are close to, and then columns are identified depending on the distance between groups of words.
3. Lastly, the table is put together based on the rows and columns detected at earlier steps.

Detecting Tables Using Lattice:

Compared to the stream technique, Lattice is more deterministic in nature. Meaning it does not rely on guesses; it first parses through tables that have defined lines between cells. Next, it can automatically parse multiple tables present on a page.

Line detection with Lattice [Fig 1]

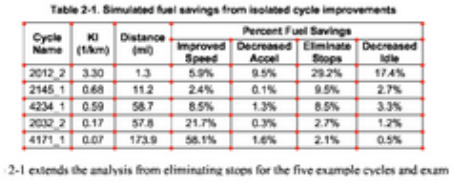


Table 2-1. Simulated fuel savings from isolated cycle improvements

Cycle Name	KI (l/km)	Distance (mi)	Percent Fuel Savings			
			Improved Speed	Decreased Accel	Eliminate Stops	Decreased Idle
2012_2	3.30	1.3	5.9%	9.5%	29.2%	17.4%
2145_1	0.68	11.2	2.4%	0.1%	9.5%	2.7%
4234_1	0.59	58.7	8.5%	1.3%	8.5%	3.3%
2032_2	0.17	57.8	21.7%	0.3%	2.7%	1.2%
4171_1	0.07	173.9	58.1%	1.6%	2.1%	0.5%

2-1 extends the analysis from eliminating stops for the five example cycles and exam

Table detection with lattice [Fig 2]

This technique essentially works by looking at the shape of polygons and identifying the text inside the table cells. This would be simple if a PDF has a feature that can

identify polygons. If it had, it would plausibly have a method to read what is inside of it. However, it does not. This is where we'll have to use a computer vision library like OpenCV to roughly perform the following steps:

1. First, the line segments are detected
2. Next, the line intersections between lines are detected by looking at the intensity of the pixels of all lines. If a pixel of a line has more intensity than the rest of the pixel, it is part of two lines and, therefore, an intersection. As shown in figure 2.
3. The edges of the table are determined by looking at the intensity of the pixels of intersected lines. Here, all the pixels of a line are taken, and the most external lines represent the boundaries of the table
4. The image analysis is translated into the PDF coordinates, where the cells are determined. Lastly, the text is assigned to a cell-based on its *x* and *y* coordinates.

Methods to Detect Tables in Scanned PDFs

It might seem impossible to identify tables in scanned images. This is because we won't be finding any text electronically present in an image; hence there cannot be a table either. This is where we'll have to use OCR and deep learning techniques to detect tables and extract all the text inside them. Now, let's look at some techniques that extract tables from PDFs that have scanned information.

Identifying Tables with Python and Computer Vision

Computer Vision (CV) is a technology that trains computers to interpret and understand the visual world. In our case of extracting tables from PDFs, we'll be using CV to help us find the borders, edges, and cells to identify tables. This is achieved by applying various filters, contours, and some mathematical operations to a PDF file. However, these techniques include some pre-processing steps on the data to perform accurately.

Now, let's actually dive into some basic python code to detect tables from scanned PDFs. Consider we have a PDF file and want to save that in an Excel sheet. We can extract normal text with OCR, but to identify tables, we'll have to use CV.

The first step we'll need to do is convert the PDF into images, and this is because most of the CV algorithms are implemented on images. As images can be turned into an array of numbers, we can find similarities between these numbers and figure out where exactly tables and text are located. Below is the code snippet:

```
from pdf2image import convert_from_path

# convert pdf file to image
images = convert_from_path('example.pdf')
for i in range(len(images)):

      # Save pages as images in the pdf
    images[i].save('page'+ str(i) +'.png, 'PNG')
```

Say, our first page is named `page_1`, we'll have to first load it into a python variable and then apply all the operations that help us identify the features of the table. Below is the complete code snippet:

```
# import cv2

import cv2
```

```
    # load the image

    file = r'page_1.png'
    table_image_contour = cv2.imread(file, 0)
    table_image = cv2.imread(file)

    # Inverse Image Thresholding
    ret, thresh_value = cv2.threshold(
        table_image_contour, 180, 255, cv2.THRESH_BINARY_INV)

    # Dilation
    kernel = np.ones((5,5),np.uint8)
    dilated_value = cv2.dilate(thresh_value,kernel,iterations = 1)

    contours, hierarchy = cv2.findContours(
        dilated_value, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

    for cnt in contours:
        x, y, w, h = cv2.boundingRect(cnt)
        # bounding the images
        if y < 50:
            table_image = cv2.rectangle(table_image, (x, y), (x + w, y -

    plt.imshow(table_image)
    plt.show()
    cv2.namedWindow('detecttable', cv2.WINDOW_NORMAL)
```

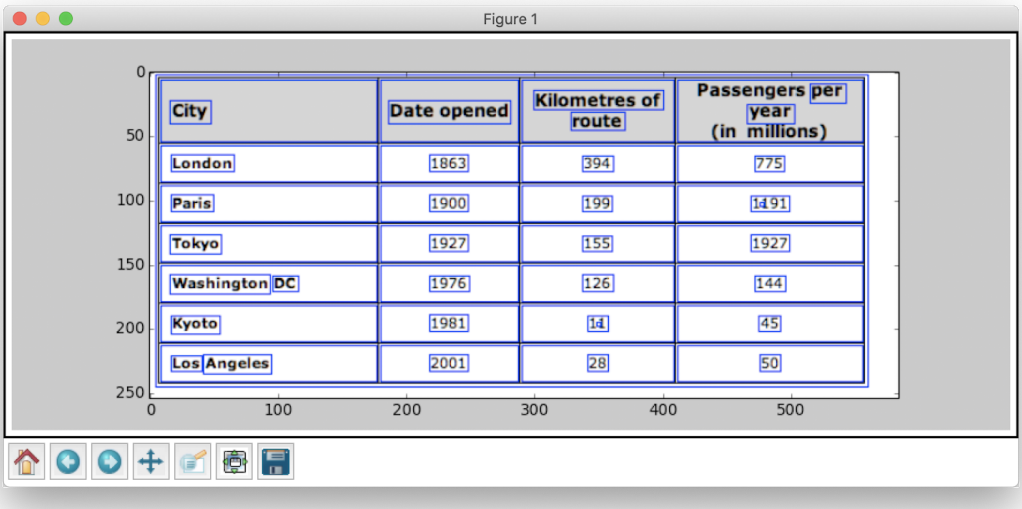
In the above code snippet, we've done a lot! Now let's try to decode this process.

First, we've imported cv2 (computer-vision package) into our program. This package is open-source and completely free to use. You can install it on your computer and give it a try. Next, we load a contour image by using the inbuilt 'imread' function from cv2. This contour image is the contrast version of the original image.

Next, we used inverse image thresholding and dilation technique to enhance the data that's in the given image. Once the images are enhanced, we use the

method `findContours` from cv2 to obtain the present image's contours. The findContours unpacks two values. Hence we'll add one more variable named `hierarchy` . When the images are nested, contours exude interdependence. To represent such relationships, hierarchy is used.

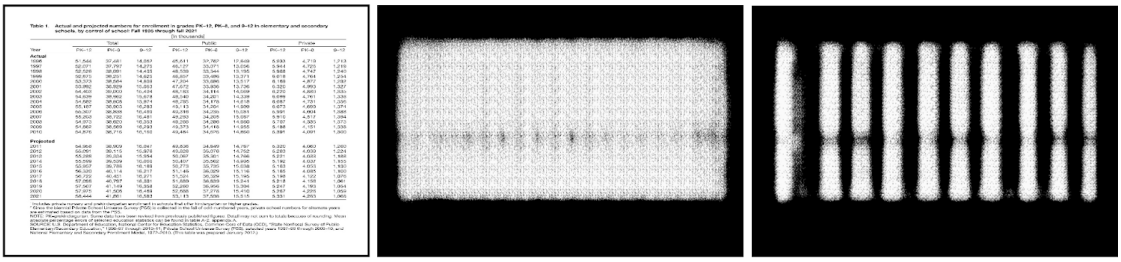
Lastly, the contours mark where exactly the data is present in the image. We iterate over the contours list that we computed in the previous step and calculate the rectangular boxes' coordinates as observed in the original image using the method `cv2.boundingRect` . In the last iteration, we put those boxes onto the original image table_image using `cv2.rectangle()` . In the end, we plot the output using matplotlib. Below is a screenshot:



Identifying Tables with Deep Learning

Deep learning had a huge impact on applications related to document understanding, information extraction, and many more. For use cases such as table extraction, many things should be considered and solid pipelines to build state of the art algorithms need to be created. In this section, we'll go through some of the steps and techniques needed to build solid neural networks to perform table extraction from a PDF file.

1. **Data Collection:** Deep-learning based approaches are data-intensive and require large volumes of training data for learning effective representations. Unfortunately, there are very few datasets like Marmot, UW3, etc for table detection and even these contain only a few hundred images. However, for documents of complicated templates and layouts, we might have to collect our own datasets.
2. **Data Preprocessing**: This step is the most common thing for any machine learning or data-science based problem. It mainly involves understanding the type of document we're working on. For example, say our goal is to export PDFs to Excel sheets. We'll have to make sure all the input data is consistent. These can be invoices, receipts, or any scanned information. But with consistency, deep learning models will be able to learn and understand the features with more accuracy.
3. **Table Row-Column Annotations**: After processing the documents, we'll have to generate annotations for all the pages in the document. These annotations are basically masks for table and column. Annotations help us to identify the tables and it's column regions from the image. Here, since all the other text inside the PDFs are already extracted by an OCR like Tesseract, only the text inside the tables have to be filtered out. Next, we'll have to define a collection of rows and multiple columns present at a horizontal level with these filtered words. However, we'll also have to consider different segmentation rules depending upon the content of a column or line demarcations, and a row can span multiple lines.
4. **Building a Model**: The Model is the heart of the deep learning algorithm. It essentially involves designing and implementing a neural network. Usually, for datasets containing scanned copies, Convolutional Neural Networks are widely employed. However, building state of the art models involves a lot of experience and experimenting. Now, let's look at some of the existing algorithms that were used to extract tables from scanned PDFs.



(a) Figure showing the raw document image. (b) Generated table mask after processing. (c) Generated column mask after processing.

Deep Learning Models generating/annotations masks from tables

Looking to export information from scanned PDFs to Excel sheets? Head over to Nanonets to automate the process of exporting from PDFs to Excel...

Get Started

Business Benefits of Automating the PDF to Excel Process

- Automation of PDFs can create and configure rules and formulas to automatically extract data from PDF to Excel. This reduces the time needed to search and copy/paste the required information manually.
- Extraction of data from images into text can be much easier by automating PDFs, using built-in OCR engines without having to type the data again manually. This reduces the probability of typos and other errors during extraction.
- Business efficiency can be improved by automating the entire extraction pipeline and running it on a batch of PDF files to get all desired information in one go. With this, we can ensure that the data is available as and when needed.
- By automating the PDFs to Excel conversion, we can easily integrate your data with any third-party software. For example, say if we want to set up an RPA process for automating invoice extraction, we could easily incorporate them with these pipelines.

Review of Some Existing Solutions for Converting PDFs to Excel

Out there, we can find several tools that can convert PDF data into Excel. However, every product has its pros and cons. In this section, we'll look at some of the free cloud/on-prem tools that we can use to convert PDF to Excel and help in automation.

Nanonets

Nanonets is an AI-based OCR software that automates data capture for intelligent document processing of invoices, receipts, ID cards and more. Nanonets uses advanced OCR, machine learning based image processing and Deep Learning techniques to extract relevant information from unstructured data. It is fast, accurate, easy to use, allows users to build custom OCR models from scratch and has some neat Zapier integrations. Digitize documents, extract data fields, and integrate with your everyday apps via APIs in a simple, intuitive interface.

How does Nanonets stand apart as an OCR software?

Pros:

- Modern UI
- Handles large volumes of documents
- Cognitive capture
- Reasonably priced
- Ease of use
- Requires no in-house team of developers
- Algorithm/models can be trained/retrained
- Great documentation & support
- Lots of customization options
- Wide choice of integration options
- Works with non-English or multiple languages
- Almost no post-processing required
- Seamless 2-way integration with numerous accounting software
- Great API for developers

Cons:

- Can't handle **very high** volume spikes
- Table capture UI can be better

EasePDF

EasePDF is an all-in-one online free PDF converter that extracts every table sheet from your PDF and saves them to Excel spreadsheets with the highest accuracy rate. It will preserve all data, layout, and formatting from the original PDF.

Pros:

- Free
- Google Drive, One Drive Integration
- Support Batch Processing
- Pre Processing Tools
- Works on Mobile Phones

Cons:

- No APIs
- Completely Cloud
- Doesn't train with custom data

pdftoexcel

pdftoexcel.com is a free online PDF to Excel conversion service that everyone can use. But for free users, the uploading might take a little bit more time.

Pros:

- Free
- Queuing for more uploads

Cons

- No APIs
- Completely Cloud
- Doesn't train with custom data
- Not so great UI
- No batch conversion on the free edition

PDFZilla

PDFZilla is a powerful tool that allows us to convert PDF documents into Excel, Word, Plain Text, Rich Text, JPG, GIF, PNG, and more file formats.

Pros:

- Good Accuracy
- Supports 20+ languages
- Supports Batch Process

Cons:

- App-only available on Windows
- Limited trial period
- Not so great UI

Adobe Acrobat PDF to Excel:

Adobe is the PDF format's original developer, so their Adobe Acrobat software should be the market-leading software. It's certainly packed with features, including the ability to convert PDF files into XL XS files for use in Excel spreadsheets. The process should be quick and painless, with the data preserved without the need for reformatting

In Adobe Acrobat, you open the PDF file you want to export, click on the Export PDF tool, choose your formats such as Excel Workbook or .xlxs, then ship. You can do this on any device, including your cell phone.

Pros

- High Accuracy
- Easily Export Features
- Comprehensive Features

Cons

- No API customisation
- Highly Prices
- Limited Trial Period

A Quick Comparison...

Feature	Nanonets	EasePDF	pdftoexcel	PDFZilla	Adobe Acrobat
User Interface	Simple, Easy, & Modern	Simple	Complicated & Unintuitive	Complicated	Simple
Integrations	Many	Limited	Limited	Limited	Many
Customization	Yes	No	No	Limited	Limited
APIs	Yes	No	No	Limited	Limited
Speed	Fast	Fast	Slow	Moderate	Fast
Accuracy	High	High	Moderate	High	High

Looking to export information from scanned PDFs to Excel sheets? Head over to Nanonets to automate the process of exporting from PDFs to Excel...

Commonly Faced Issues while Exporting PDF into Excel

1. **Finding the Right Algorithms:** When it comes to automating PDF to Excel text extraction, one common issue faced by most of the developers is finding the correct algorithm that can parse and understand the entire PDF document. Here's a question posted on StackOverflow regarding the same. This is because, out there, we find several deep learning algorithms, but again, we'll have to finetune them based on our use-case and type of data; this involves retraining the entire model with our own data using the existing model. Also, developers cannot promise the same accuracy after the models are re-trained as deep learning models require lots of hyperparameter tuning.
2. **Table Extraction:** For automating the process of converting PDFs to Excel, table extraction plays a crucial role. Hence, identifying tables and parsing through them is also critical for this automation. Most new developers might browse through different CV and deep learning-based algorithms for table extractions, which is challenging. Few of them might use services like Textract, Rossum, etc., which are complicated to integrate with different workflows of various backends.
3. **Post Processing and Additional Scripting**: Text that has been extracted from PDFs might not be accurate all time. Hence lot's of post-processing steps are followed using different techniques, for example, say if we want only particular columns in tables to be extracted or list down all the dates in a column from the PDF. In such cases, OCR isn't enough. We'll have to rely on techniques using regular expressions, language models, and simple nested conditions. This is again some additional work. Hence, we'll have to be careful when choosing a powerful extraction algorithm that comprises most of the post-processing.

Conclusion

In this article, we've seen how we can export PDF into Excel using various techniques. We've reviewed in depth how we can use OCR and Deep Learning for converting electronic and non-electronic PDFs into an Excel document.

Further, we've seen how we can build a solid pipeline using deep learning models for PDF to Excel conversion. Lastly, we've seen some of the popular tools that we can directly use to build workflows.

You might be interested in our latest posts on:

- AWS Textract
- Data Extraction
- Best OCR Software
- PDF to Excel
- BPO Automation
- Invoice Processing
- Fuzzy Matching
- Fuzzy Logic
- Google Cloud Vision
- Invoice Management
- Purchase Order Matching or PO Matching
- Three-way Matching
- Payment Reconciliation
- AP Automation
- PDF to JSON