Automated

Invoice

Processing

System

Final Report - GAIN Team AA

July 31, 2024

* *Asheer Mogal, Sanjana Konka, Aditya Samyak Gadepalli*

# 

# 

# 

# List of Contents

* Problem statement
* Where the solution lies
* Trial and error
* The working prototype
* Looking ahead
* Resources used
* References

## 

## Problem Statement

Our team was tasked to automate invoices as part of the GAIN internship. We were to create a working prototype of an invoice reader primarily using open-source software. Global Alliant wanted to reduce human interference in document processing, and wanted data extracted from invoices on a high level. The invoice reader has to detect and extract some key variables from the invoice document. These are as follows:

- Vendor Name

- Vendor Contact Information

- Vendor Contact Person

- Invoice Number

- Invoice Date

- Due Date

- Total Invoice Amount

- Line Item Description

- Line Item Quantity

- Line Item Amount

- Contract Start Date

- Contract End Date

## Where the Solution Lies

From the beginning, it was clear machine learning was going to be the crux of our project. Invoice readers and processing systems use technologies like OCR (Optical Character Recognition) and NLP (natural language processing) to extract data from the invoices. There are 4 key steps to an invoice reader.

**Data Capture**:

* **Scanning**: Physical invoices are scanned and converted into digital format. All invoices accepted had to be in pdf form.
* **OCR**: Optical Character Recognition technology reads and extracts text from digital images of invoices.
* **Image Preprocessing**: Enhances the quality of scanned images to improve data extraction accuracy.

**Data Extraction**:

* **Field Identification**: The software identifies and extracts key data variables, such as the invoice number, invoice date, contract start date, etc.
* **Machine Learning**: Algorithms learn to recognize various formats and layouts of invoices from different suppliers.

**Reporting and Analytics**:

* Provides analytics on invoice processing, spending patterns, and supplier performance.

We worked closely with the team tasked with HCD development, and an outside website shell to hold the invoice reader. Our teams helped each other learn more about the project and collaborated on various efforts such as finding the target users and what they would want, and understanding the mindsets of users and what features they would like in the website. Key features discussed about the invoice reader included:

1. Document Upload: A file upload interface where users can upload files (folders, PDFs, or photos).

2. Output Display: A space where the variables that were extracted are shown.

3. Integration of OCR and NLP: OCR is used to extract text from invoices, while NLP is used to pinpoint the appropriate variables.

4. Data Storage: Keeping the data that has been extracted in a database.

5. History Section: A section where previous bills can be viewed and removed, but not edited.

## 

## 

## Trial and Error

In a nutshell, trial and error was the bread and butter of implementation during our internship. This process fosters creativity and innovation as each error provides valuable insights, leading to a better understanding of the problem and progressively closer to a viable solution. Trial and error is essential for testing hypotheses, optimizing processes, and developing new technologies.

### Plan One - Basic OCR implementation

Our first initial plan was to use synthetic data and create synthetic invoices, to feed the machine and train it to read invoices. We were going to use Open CV to also enhance the image and make it more readable. The steps were:

1. **Populate Data**: Use scripts to automatically fill templates with synthetic data, such as random company names, invoice numbers, dates, item descriptions, quantities, etc.
2. **Diversify Data**: Ensure the synthetic invoices represent a wide range of scenarios, including different currencies, tax rates, and address structures.
3. **Image Processing**: Utilize OpenCV (Open Source Computer Vision Library) to preprocess the generated invoice images.

* **Increase Brightness**: Adjust the brightness levels to improve the visibility of text and other elements.
* **Change Font**: Apply different font styles and sizes to simulate various real-world invoices and test the OCR system.

1. **OCR Training**: Use the enhanced synthetic invoices to train an OCR using machine learning algorithms or deep learning frameworks (e.g., Tesseract, TensorFlow).
2. **Evaluation and Tuning**: Continuously evaluate the model's performance using a validation set of synthetic invoices. Fine-tune the model by adjusting parameters and augmenting the training data as needed.

# Code for setting up Tesseract OCR

!apt-get install tesseract-ocr

import pytesseract

from PIL import Image

def extract\_text\_from\_image(image\_path):

image = Image.open(image\_path)

text = pytesseract.image\_to\_string(image)

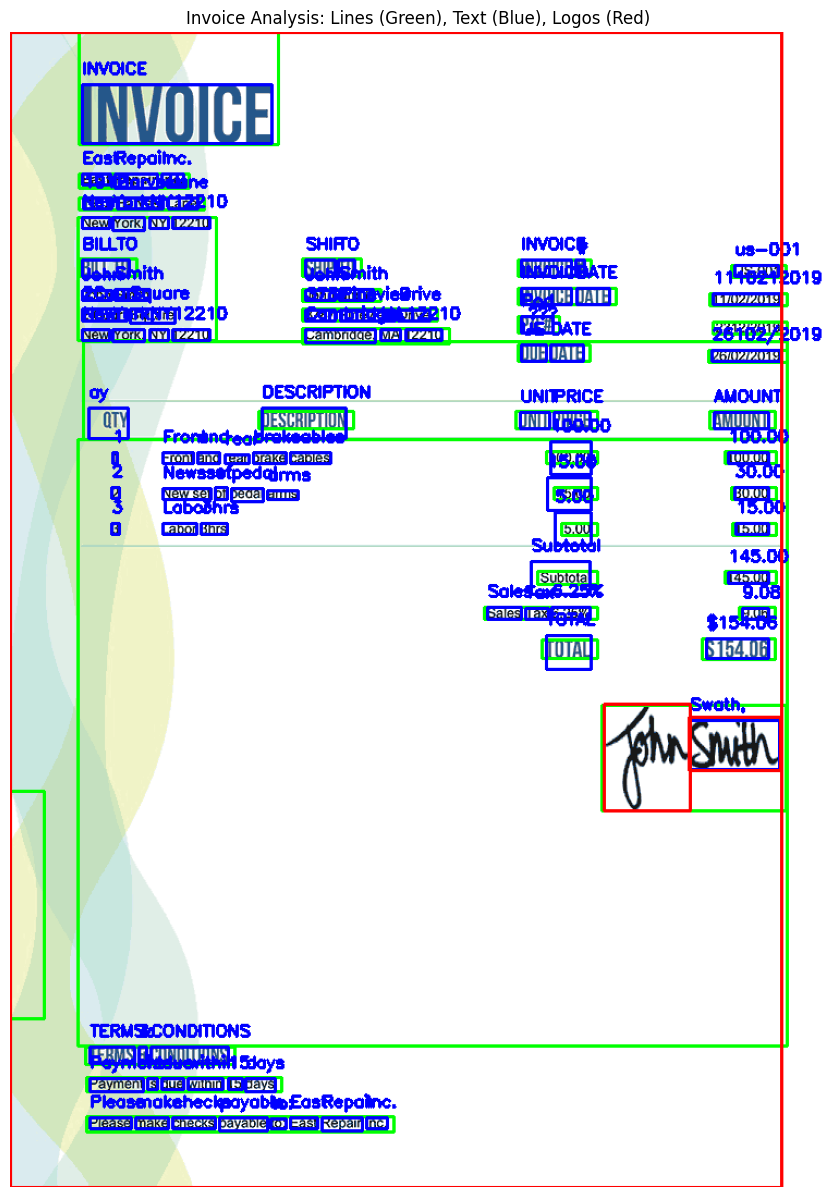
return text

# Example usage

image\_path = 'path\_to\_invoice\_image.jpg'

extracted\_text = extract\_text\_from\_image(image\_path)

print(extracted\_text)



Unfortunately, this method proved to not be viable, mostly because of the limitations we had with finding a synthetic invoice generator. The method also lacked the capability to understand complex invoices and didn’t extract data well, resulting in low confidence intervals.

### Plan Two - LayoutLMv3

To address the problems of the first approach, we targeted using the language model “Layout Language Model 3” or LayoutLmv3 for short. LayoutLMv3 from Hugging Face represents a significant advancement in document understanding by combining textual, visual, and spatial layout information into a unified transformer-based architecture. This sophisticated integration allows the model to effectively interpret and extract structured data from complex documents, enhancing tasks such as form recognition, invoice processing, and information retrieval with a high degree of accuracy and contextual awareness. With the help of the FUNSD dataset, we planned to train the model.

While this approach showed a great increase in the accuracy and the ability to understand more complex invoice layouts, it lacked the ability to learn and “train”. An essential part of machine learning is the ability of the model to adapt and train when corrected, also known as reinforcement learning. Another liability was the FUNSD dataset, which had a maximum of 199 rows which meant it was small and relatively insignificant in the fine-tuning process. A larger dataset would have been ideal, but unfortunately, we could not find any open-source datasets to suit our needs in our research.

Some further enhancements were made to the implementation of the LayoutLMv3 model. We continued to train it and add validation steps in an effort to increase accuracy and efficiency but to no avail. This method slowly started to also require significant computational power, something we didn’t have and could not afford with the budget of the project. So we decided

**Model Initialization and Configuration:**

from transformers import LayoutLMv3Processor, LayoutLMv3ForTokenClassification

import torch

processor = LayoutLMv3Processor.from\_pretrained("microsoft/layoutlmv3-base")

model = LayoutLMv3ForTokenClassification.from\_pretrained("microsoft/layoutlmv3-base")

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

model.to(device)



**Data Preparation:**

from datasets import load\_dataset

dataset = load\_dataset("funsd", split="train")



**Training Loop:**

from torch.optim import AdamW

from transformers import get\_scheduler

optimizer = AdamW(model.parameters(), lr=5e-5)

num\_training\_steps = len(dataset) \* 3

lr\_scheduler = get\_scheduler(

"linear",

optimizer=optimizer,

num\_warmup\_steps=0,

num\_training\_steps=num\_training\_steps

)

model.train()

for epoch in range(3):

for batch in dataset:

inputs = processor(batch["image"], return\_tensors="pt").to(device)

labels = batch["labels"].to(device)

outputs = model(\*\*inputs, labels=labels)

loss = outputs.loss

loss.backward()

optimizer.step()

lr\_scheduler.step()

optimizer.zero\_grad()



```

### Plan Three - Dolly v2

The next phase of the project shifted to the model Dolly V2. Dolly V2, developed by Databricks, is an advanced large language model built on open-source datasets, designed to facilitate a range of natural language processing tasks with high efficiency and accuracy. Using transformer architecture, Dolly V2 incorporates pre-training on diverse text corpora, allowing it to understand and generate human-like text. Its training process emphasizes scalability and fine-tuning. Making it a prime pick for our needs. Furthermore, its open-source nature encouraged continuous work and improvements and kept our budget in check. We decided to use the ‘invoices-and-receipts\_ocr\_v1’ dataset, containing 2300 rows of data. This dataset is a comprehensive collection designed for training and evaluating OCR systems specifically tailored for processing invoices and receipts.

**Model Initialization:**

from transformers import DollyV2ForSequenceClassification, DollyV2Tokenizer

model\_name = "dolly-v2-3b"

tokenizer = DollyV2Tokenizer.from\_pretrained(model\_name)

model = DollyV2ForSequenceClassification.from\_pretrained(model\_name)

model.to(device)



**Data Preparation:**

from datasets import load\_dataset

dataset = load\_dataset("mychen76/invoices-and-receipts\_ocr\_v1", split="train")



**Training Loop:**

from transformers import AdamW, get\_scheduler

optimizer = AdamW(model.parameters(), lr=5e-5)

num\_training\_steps = len(dataset) \* 3

lr\_scheduler = get\_scheduler(

"linear",

optimizer=optimizer,

num\_warmup\_steps=0,

num\_training\_steps=num\_training\_steps

)

model.train()

for epoch in range(3):

for batch in dataset:

inputs = tokenizer(batch["text"], return\_tensors="pt", padding=True, truncation=True).to(device)

labels = batch["labels"].to(device)

outputs = model(\*\*inputs, labels=labels)

loss = outputs.loss

loss.backward()

optimizer.step()

lr\_scheduler.step()

optimizer.zero\_grad()



The big issue surrounding the Dolly V2 was its monumental computation need. We didn’t have the resources to continue working on the Dolly model because it constantly needed a lot of power, and as a result, we had to quit on this iteration as well. We also noticed that the results were similar to the LayoutLMv3, so the minuscule increase in the accuracy of data extraction didn’t warrant such a huge jump in computational power and money.

## The Working Prototype - Mistral 7B

All of the trials led to the final solution to our initial problem. We found the Mistral 7B model. Mistral7B is a high-performance language model developed by Mistral AI, with 7 billion parameters, making it powerful enough to handle a wide range of natural language processing tasks with efficiency. It utilizes transformer architecture to deliver results in text generation, comprehension, and translation, benefiting from its training on diverse datasets. We decided to continue using the much refined and helpful `invoices-and-receipts\_ocr\_v1` dataset with the same aforementioned 2300 rows of data. Fortunately, this model had everything we were looking for. It was extremely powerful while having the right balance of not needing too much power while being extremely accurate. The one thing remaining was to fine-tune it.

The final step in the project was to enhance and fine-tune the Mistral7B model's ability to not only extract text but also convert it into a structured JSON format. This method proved to be the best solution for converting scanned invoices into an easily manageable and accessible format, automating the end-to-end process of invoice data processing. We also used Low-Rank Adaption (Lo-Ra) to fine-tune the model for better performance. Lo-Ra is a technique employed to fine-tune large language models, like Mistral7B, for enhanced performance with lower computational overhead. By decomposing the weight updates into low-rank matrices, LoRA allows for efficient adaptation of the model to specific tasks without the need to retrain the entire network. This approach not only reduces the memory and computational requirements but also accelerates the fine-tuning process, making it feasible to customize large models for various applications. Furthermore, we used a specialized dataset of invoices and receipts from hugging face to train and enhance the model.

**Model Initialization:**

from transformers import Mistral7BForSequenceClassification, Mistral7BTokenizer

model\_name = "mistral7b"

tokenizer = Mistral7BTokenizer.from\_pretrained(model\_name)

model = Mistral7BForSequenceClassification.from\_pretrained(model\_name)

model.to(device)



**Data Preparation:**

from datasets import load\_dataset

dataset = load\_dataset("mychen76/invoices-and-receipts\_ocr\_v1", split="train")



**Training Loop:**

from transformers import AdamW, get\_scheduler

optimizer = AdamW(model.parameters(), lr=5e-5)

num\_training\_steps = len(dataset) \* 3

lr\_scheduler = get\_scheduler(

"linear",

optimizer=optimizer,

num\_warmup\_steps=0,

num\_training\_steps=num\_training\_steps

)

model.train()

for epoch in range(3):

for batch in dataset:

inputs = tokenizer(batch["text"], return\_tensors="pt", padding=True, truncation=True).to(device)

labels = batch["labels"].to(device)

outputs = model(\*\*inputs, labels=labels)

loss = outputs.loss

loss.backward()

optimizer.step()

lr\_scheduler.step()

optimizer.zero\_grad()

### 

## Looking Ahead

While the Mistral 7B is a great start and a fully functional prototype, it’s imperfect. It’s A solution to the initial problem statement at hand, and there is still work to be done to fine-tune it even better. Additionally, there are further steps to take in this project to level up the invoice reader even more.

1. Integrating the front and back-end:

The first step is to connect the website from the HCD team to the model we developed. Both are successful prototypes, that now need to be implemented together to experiment and enhance further. Connecting the two will finally help the model also utilize all of the front-end features developed, such as user authentication, role-based access control, data encryption, etc.

1. Integrating to a database:

A database like Snowflake would be ideal for this. Snowflake is a cloud-based data warehousing platform that provides storage and analytics solutions with a fully managed service. Additionally, its integration with various machine learning tools and frameworks helps with data access and model training. The database will also help with saving the invoice for up to 30 days in the history tab we have planned for the website.

1. Confidence scores:

“Confidence scores” derived from the confidence interval will help determine if the model did a good job reading the invoice or if it needs to do it again. An assigned “confidence score” is not exactly the confidence interval percentage, so finding the correlation between those two will be the key to extracting this vital information for the user. Files that have low “confidence scores” will either be prompted to upload again, or have the model run again on the same invoice.

## Resources used

* **Google Colab (Python):** For developing and testing AI models.
* **OpenCV, TensorFlow, Hugging Face Transformers:** For data processing and model fine-tuning.
* **OCR and NLP Techniques:** For text extraction and data interpretation.
* **Amazon Web Services (AWS):** For the use of the ‘EC2 instance’ to compute in the cloud.
* **Datasets used:** 
  + FUNSD Dataset - <https://guillaumejaume.github.io/FUNSD/>
  + Invoices-and-receipts\_ocr\_v1Dataset <https://huggingface.co/datasets/mychen76/invoices-and-receipts_ocr_v1>
* **Special Acknowledgments** to *Ethan Ford* and *Sangavarapu Ritvic Kashyap* for helping and guiding throughout the whole project.

## References

1. <https://www.analyticsvidhya.com/blog/2021/06/data-extraction-from-unstructured-pdfs/>
2. <https://youtu.be/el9MRSgmz1s>
3. <https://stackoverflow.com/questions/50829874/how-to-find-table-like-structure-in-image>
4. <https://www.kaggle.com/discussions/general/263960>
5. LayoutLM and its versions: <https://www.klippa.com/en/blog/information/layoutlm-explained/>
6. <https://huggingface.co/databricks/dolly-v2-3b>