

# Document AI - Invoice Parser

## Unstructured Data -> Structured Data

### Introduction

With the growth of digitalization in all sorts of industries, companies want to automate most of their manual work, as manual work is prone to errors and is time consuming. One such case is the manual entry of form-like documents for further tasks, for example reading an invoice and filling entries like invoice date, seller name, buyer name, etc manually. This created a need for an algorithm that can automate this task by just taking invoice pdf as an input and outputting a json file containing all the extracted fields from that invoice.

**Supplier name** Examplewebshop.com

**Recipient** Klippa  
Purchasing department  
Lübeckweg 2  
9723HE Groningen  
The Netherlands  
info@klippa.com

**Payment method** Payment method:  
Paid with Paypal

**Supplier contact information**  
Spijkerkade 3  
1021 JS Amsterdam  
The Netherlands  
Phone: 0203301924  
Email: info@examplewebshop.com  
Website: www.examplewebshop.com  
BIC: NL24RABO0110562482  
IBAN: NL441231 94434231  
VAT Number: NL453534420001

**Weborder 10000734**

**Invoice** #18800000  
Invoice date: 21-12-2018

**Line items**

| Count | SKU       | Description       | Price  | VAT% | Net. amount | VAT    | Total amount |
|-------|-----------|-------------------|--------|------|-------------|--------|--------------|
| 76    | SKU-21371 | Example items     | € 0.59 | 21%  | € 37.06     | € 7.78 | € 44.84      |
| 24    | SKU-10465 | Replacement parts | € 0.59 | 21%  | € 11.79     | € 2.48 | € 14.18      |
| 1     |           | Shipping fee      | € 3.95 | 21%  | € 3.26      | € 0.69 | € 3.95       |
| 1     |           | Transaction fee   | € 0.00 | 21%  | € 0.00      | € 0.00 | € 0.00       |

**Invoice total, VAT total**

|                            |          |              |
|----------------------------|----------|--------------|
| Total netto amount         | €        | 53.02        |
| VAT 21%                    | €        | 10.93        |
| <b>Total including VAT</b> | <b>€</b> | <b>62.95</b> |

Sample Output


## Methods

Several methods were tried and researched on, so as to compare the performance and accuracy of each approach in order to find the best approach.

### Method 1 : Easy OCR and spacy

#### Easy OCR :

Easy OCR is a python's OCR library used for text extraction from images, we used this library to convert our image into a text format. Firstly, the images were preprocessed (Grayscale, Gaussian Blur, Binary Thresholding). Then, the Easy OCR was used on the preprocessed images to extract the text. This extracted text was further used in the NER (Named Entity Recognition) model.

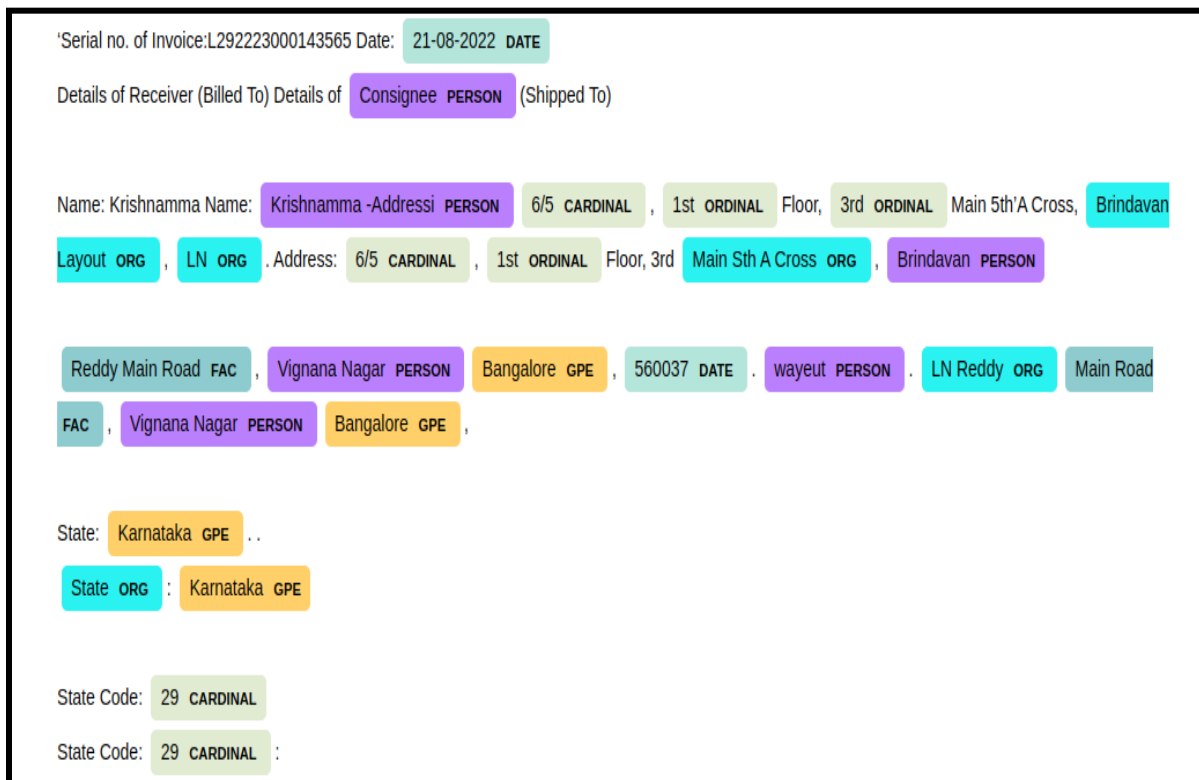
|   |          |  |       |   |          |       |           |
|---|----------|--|-------|---|----------|-------|-----------|
| <b>DK SOFTWARES AND SERVICES</b><br>687 SAHEED NAGAR BHUBANESWAR<br>GSTIN NO-21EFKPR083F129<br>State: 21-Orissa<br>GSTIN: 21EFKPR083F129  |          | Invoice No: TAXI003353<br>Dated: 05/12/2022  |       |   |          |       |           |
| <b>Bill To :</b><br>TALLY SOLUTIONS PVT LTD-BANGALORE<br>23 &mp; 24 AMR TECH PARK II HONGASANDRA<br>HONGASANDRA MAIN ROAD BANGALORE<br>GSTIN-29AAACP7879D120<br>State: 29-Karnataka<br>GSTIN: 29AAACP7879D120                               |          | Delivery Note No: 000369<br>Dated: 05/12/2022<br>No. of Packs:<br>Other Reference(s):  |       |   |          |       |           |
| <b>Ship To :</b><br>TALLY SOLUTIONS PVT LTD-BANGALORE<br>23 &mp; 24 AMR TECH PARK II HONGASANDRA<br>HONGASANDRA MAIN ROAD BANGALORE<br>GSTIN-29AAACP7879D120<br>BANGALORE 560088 Karnataka<br>State: 29-Karnataka<br>GSTIN: 29AAACP7879D120 |          | Despatch Dec. No.:<br>Dated:<br>Despatched Through:<br>Destination: BANGALORE-Karnat<br>Bla<br>Agent:<br>Del. Inst.:<br>Payment Due Date: 05/12/2022 |       |   |          |       |           |
| Sr. No.   | HSN Code | Particulars  | Tax%  | Qty   | Rate     | Disc% | Amount    |
| 1   | 998297   | Reimbursement Expenses-Event<br>Business Support Service   | 18.00 | 1 Nos   | 20435.00 |       | 20435.00  |
| IGST  |          |  |       |   | 18.00%   |       | 3,678.30  |
| Round Off   |          |  |       |   |          |       | -0.30     |
| Total   |          |  |       | 1 Nos   |          |       | 24,113.00 |
| Amount(in words): Twenty Four Thousand One Hundred Thirteen INR Only  |          |  |       |   |          |       |           |
| <b>Account Details</b><br>DK SOFTWARES AND SERVICES<br>PUNJAB ANANATHAL BANK<br>A/C NO- 7676002100001168<br>IFSC: PUNB0767600<br>BRANCH: KANAKAPUR<br>CUTTACK (ODISHA)  |          |  |       |   |          |       |           |
|   |          |  |       | <br>For, DK SOFTWARES AND SERVICES<br>Authorised Signatory |          |       |           |

Easy OCR Output

---

## Spacy :

Spacy is a python's library used for NER (Named Entity Recognition). Spacy takes text as an input, tokenizes the text and then predicts some common fields like, PERSON, ORG, DATE, etc from that text.



Spacy Output

Spacy has several pre-trained models to choose from like, small model, large model and the RoBERTa model. We chose the uncased models for small and large and compared the accuracy. Since the data in an invoice document is quite limited, the small model's accuracy came out to be slightly greater than that of the large model.

---

## Results :

We tried and tested this model on some invoices to check the efficiency of this approach, it worked fine on many invoices for the fields like invoice date, but was not that effective for other name fields.

```
2) 23.jpeg

Invoice Date : 4/11/2023
Company Name : PIN

Using CPU. Note: This module is much faster with a GPU.
Using CPU. Note: This module is much faster with a GPU.

3) 28.jpeg

Invoice Date : 9902016147
Company Name : OBAL Group GLOBAL SOFTWARE
```

Spacy Inference

## Findings :

- Easy OCR is a library that works better with the GPU and is a bit slow on using the CPU alone. It took around 15 seconds to extract the text from each image.
- Spacy is used in NLP tasks and works better if the text is semantic and holds a meaning, whereas in our use case, text is in the form of key : value pairs. Due to this reason, this approach failed in most of the invoices.
- Spacy gave a good result for the date field, but failed drastically on other fields like buyer name, seller name, etc.
- This approach is a purely NLP solution and does not take the positions of text into account.

---

## Method 2 : Tesseract OCR

Tesseract OCR is Google's OCR engine and PyTesseract is the python's wrapper for this engine that can be used directly in our python code.

Tesseract OCR works similar to the Easy OCR, takes an image as an input and returns the output, but its output has a variety of fields and key information. We can extract the output in many forms like string, dictionary and dataframe. We extracted the text in the form of a dataframe. Output had various features like the text itself, bounding boxes, para number, block number, line number, word number, etc.

Out[75]:

|     | level | page_num | block_num | par_num | line_num | word_num | left | top  | width | height | conf      | text       |
|-----|-------|----------|-----------|---------|----------|----------|------|------|-------|--------|-----------|------------|
| 4   | 5     | 1        | 1         | 1       | 1        | 1        | 654  | 125  | 44    | 26     | 90.456886 | OK)        |
| 5   | 5     | 1        | 1         | 1       | 1        | 2        | 708  | 130  | 179   | 16     | 78.731796 | SOFTWARES  |
| 6   | 5     | 1        | 1         | 1       | 1        | 3        | 896  | 125  | 67    | 26     | 78.731796 | AND        |
| 7   | 5     | 1        | 1         | 1       | 1        | 4        | 974  | 130  | 140   | 16     | 95.575264 | SERVICES   |
| 9   | 5     | 1        | 1         | 1       | 2        | 1        | 661  | 166  | 50    | 37     | 81.337814 | 684        |
| ... | ...   | ...      | ...       | ...     | ...      | ...      | ...  | ...  | ...   | ...    | ...       | ...        |
| 127 | 5     | 1        | 9         | 1       | 3        | 1        | 142  | 1981 | 114   | 23     | 76.816223 | BRANCH)    |
| 128 | 5     | 1        | 9         | 1       | 3        | 2        | 262  | 1981 | 158   | 26     | 66.789734 | KANTAPADA) |
| 130 | 5     | 1        | 9         | 1       | 4        | 1        | 138  | 2008 | 124   | 38     | 35.729332 | CUTTACR)   |
| 131 | 5     | 1        | 9         | 1       | 4        | 2        | 271  | 2008 | 114   | 38     | 65.808868 | (ODISHA)   |
| 135 | 5     | 1        | 10        | 1       | 1        | 1        | 111  | 43   | 1534  | 2071   | 95.000000 |            |

86 rows × 12 columns

Tesseract Output

Now for testing this approach, we started with just a single field, i.e, InvoiceDate field. We pre processed this dataframe to extract all the rows having a datetime object in it.

- We first combined the text tokens extracted on the basis of block number and the distance between them.
- Then created n-grams for the combined texts.
- Used datetime parser to find fields having datetime objects in them.

---

Now that the fields having dates were extracted, it was required to add some neighbor information to train the model. For example if a date field has "invoice" written in its neighboring location, then it is surely the invoice date.

Firstly we took just a few neighbor fields like "invoice date", "date", "dated", "bill date" etc, and annotated them 1 or 0 based upon their presence in the top or left region of the concerned field.

Then this final dataset was given labeling as "isInvoiceDate" or "notInvoiceDate" manually. The final dataset had 36 rows and 19 columns.

Now different model fittings were tested on this dataset for comparing performance.

## Model 1 : Keras Sequential ANN Model

A simple ANN model was trained on this dataset for 100 epochs.

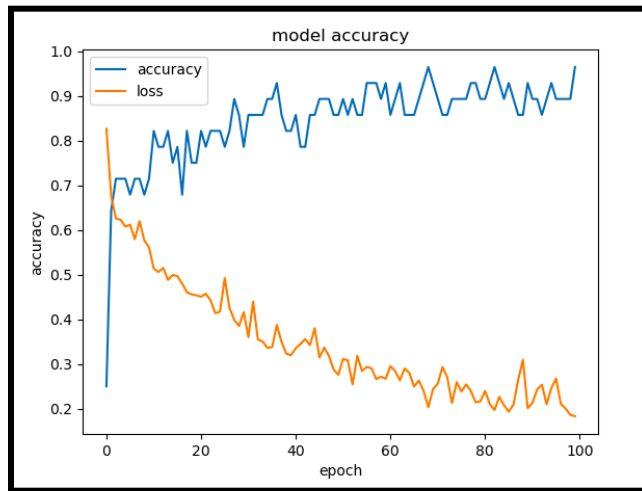
```
In [18]: model.summary()
```

```
Model: "sequential"
```

| Layer (type)    | Output Shape | Param # |
|-----------------|--------------|---------|
| dense (Dense)   | (None, 64)   | 1216    |
| dense_1 (Dense) | (None, 32)   | 2080    |
| dense_2 (Dense) | (None, 1)    | 33      |

```
=====  
Total params: 3,329  
Trainable params: 3,329  
Non-trainable params: 0  
=====
```

Keras Model Summary

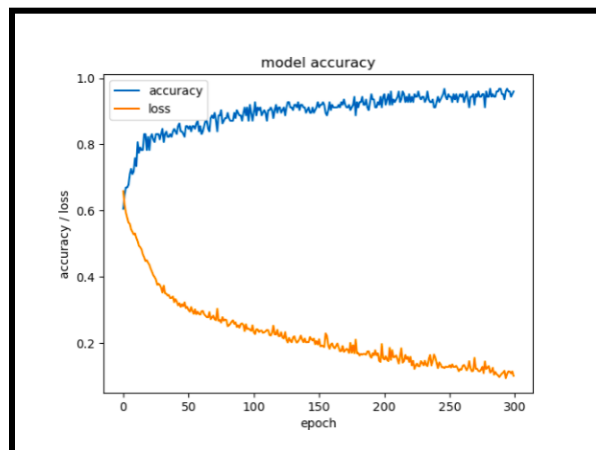


Keras Model Training

This model gave a testing accuracy of 87.5% and predicted 7 out of 8 testing images correctly. On increasing the dataset by 20 invoices and fine tuning the model, we were able to reach an accuracy of 88%

On testing the correlation between fields, we found out that neighbor fields were giving a really low correlation, so we changed neighbor extraction technique, and rather than taking its presence (1/0), we took its distance from the concerned field.

Finally the accuracy came out to be 93.55%



Keras Model Training

---

Now some performance benchmarks were performed on this model.

- Output was as follows

```
Max Time Taken : 7.416690826416016
Min Time Taken : 0.6528520584106445
Avg Time Taken : 3.2954273043938405
```

- A benchmark was prepared for monitoring CPU and RAM usage.

```
In [68]: inferenceTime, maxCPU, minCPU, maxMemory, minMemory = benchmark()
clear_output(wait=True)
print("TIME TAKEN : ", inferenceTime, "SEC")
print("MAX CPU : ", maxCPU, "%")
print("MAX MEMORY : ", maxMemory//1024*1024, "MB")
print("MIN MEMORY : ", minMemory//1024*1024, "MB")
TIME TAKEN : 5.411151170730591 SEC
MAX CPU : 9.36 %
MAX MEMORY : 750.0 MB
MIN MEMORY : 625.0 MB
```

## Keras Model Benchmark

### Model 2 : XGBOOST Model

XGBoost means extreme gradient boosting and it is an ensemble learning method that uses ensembles of decision trees to train the model. On applying the XGBoost model and comparing the accuracy with ANN model, we found that XGBoost was a clear cut winner in terms of accuracy. But the base model was still not generalizing good on the unseen data, so we performed some hyperparameter tuning using techniques like GridSearchCV but the generalization still remained bad. The problem was the low correlation between fields.

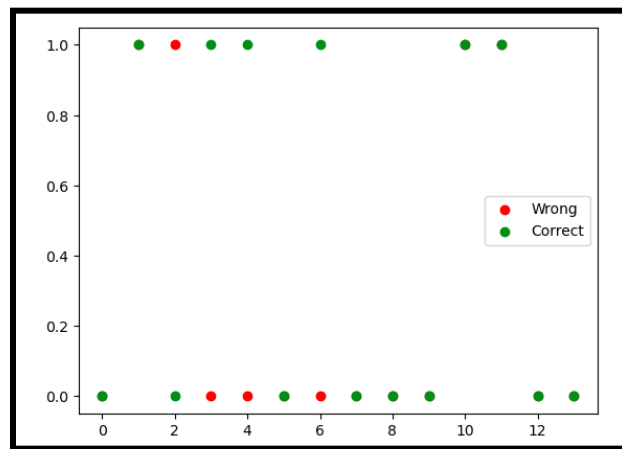
We tried adding more and more logical fields so as to improve correlation and hence the accuracy. We added a few context keywords like "invoiceDate" and used the distance of these keywords from the concerned field. Correlation improved but still it was not satisfactory. Now, rather than taking context keywords' distance, we took context keyword's X and Y positions for training, fine tuned the XGBoost model and tried to improve the accuracy.

Since the dataset was really small, too many context keywords reduced the correlation significantly, so on trying with just two keywords, we achieved an 89% accuracy with just the base model, i.e, without hyperparameter tuning.



---

Now, as we were using many columns for neighbor fields, there was a need to combine these multiple columns and encode them into a single column for training. So we tried some encoding techniques and finally achieved a correlation of ~30% and an accuracy of 82%.



XGBoost Model Output

### Findings :

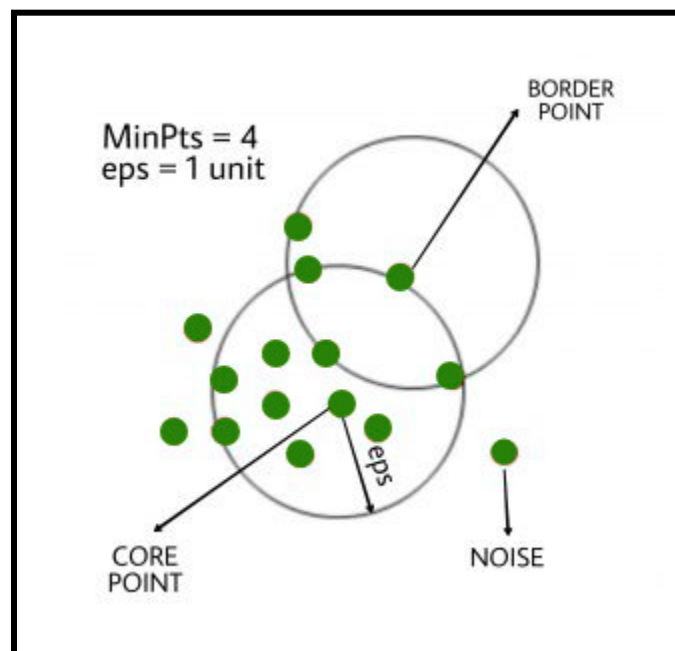
- One of the major problems with this approach was the OCR. Tesseract OCR was only 50% efficient in extracting the text correctly, since tesseract OCR is better for semantic texts that hold a meaning.
- There was a need for a Zonal OCR technique which could read text in the form of zones.

---

## Method 3 : Clustering

Clustering is an unsupervised learning mechanism that divides the data points into several groups based upon some parameters. There are several clustering algorithms like K-Means, K-Nearest-Neighbours, etc. But, these algorithms are good for spherical clusters and when the number of clusters to be made are already known. Since in our use case, each invoice can be different and is not template based, the number of clusters can differ in each invoice, so these algorithms won't work for us.

We here used an algorithm called DBSCAN (Density Based Spatial Clustering For Applications With Noise). In this algorithm the number of clusters need not to be defined and just the data points and two hyperparameters namely epsilon( $\epsilon$ ) and minimum samples( $\text{min\_samples}$ ) have to be provided for the clustering job.



DBSCAN Algorithm

Eps : Epsilon defines the distance after which the data points are classified to be other clusters. In simple words, all the points within the eps radius of a point are considered to be the same cluster.

Min-Samples : Min-Samples defines the number of data points required in an Eps radius to be classified as a cluster. All the data points which fail this property are known as noise.

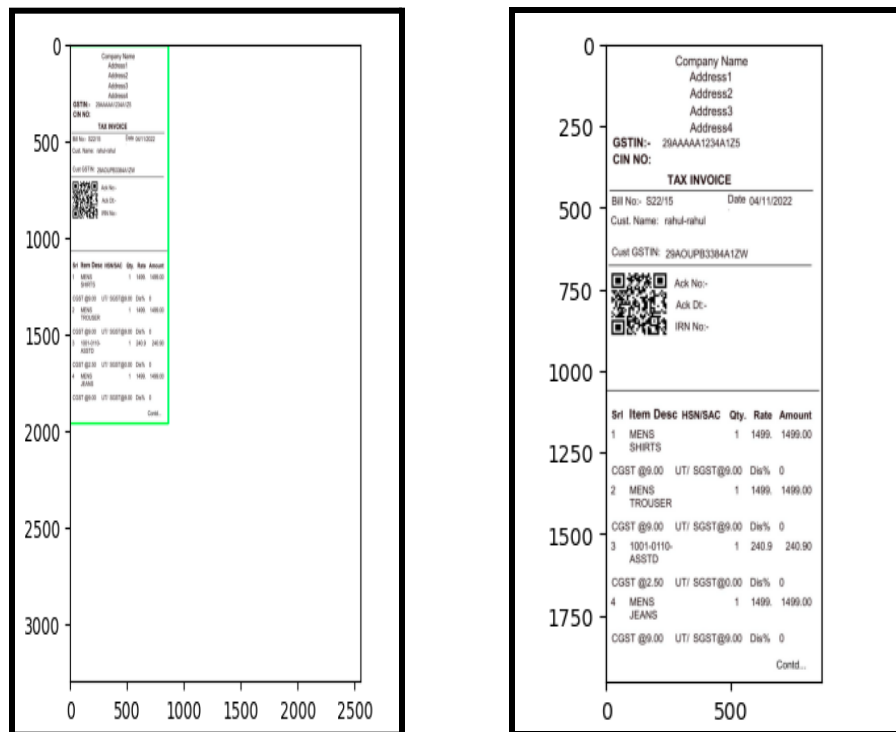
For using this algorithm, we pre processed the image so as to normalize the output by DBSCAN.

## Image Pre Processing :

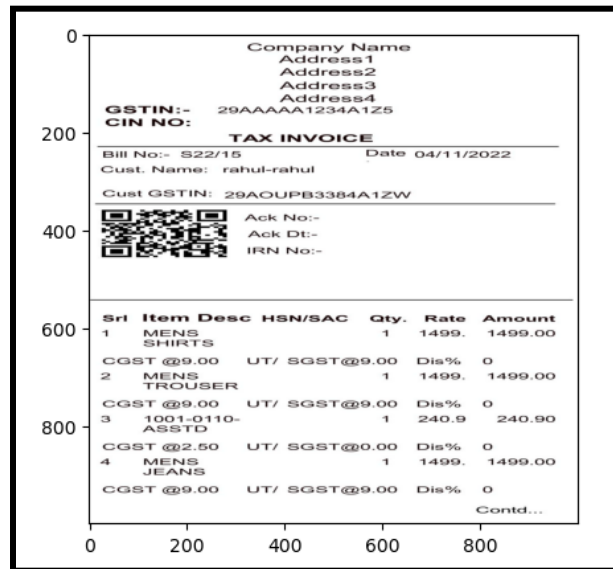
It included :

- Cropping the images for just the relevant area containing text.

Example :

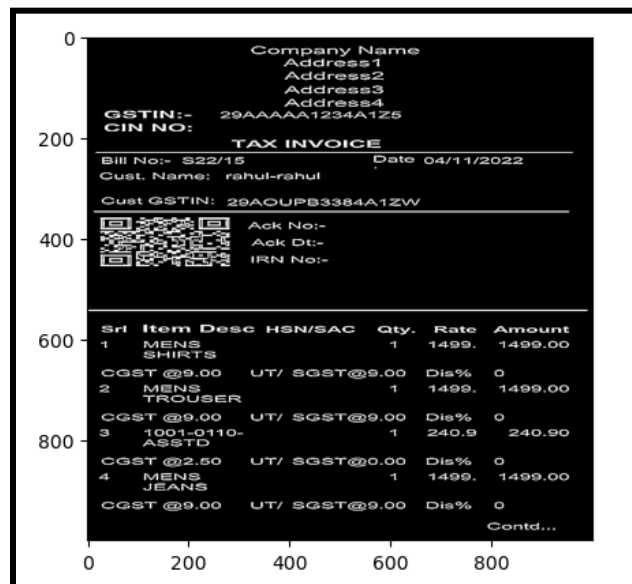


- Converting images to a fixed size of 1000px X 1000px



After Resizing to 1000x1000

- Converting images to grayscale and adding blur and binary thresholding to increase contrast.



After applying Gray/Blur/Thresh

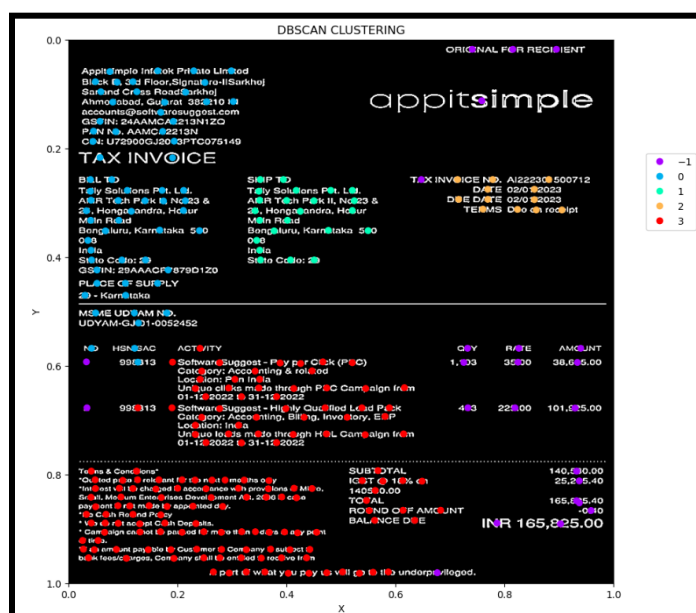
## Clustering :

- These pre-processed images were passed through the tesseract OCR to extract all the text tokens from the image.
- The tesseract output was pre processed.
- DBSCAN clustering was applied with random parameters.

```
In [16]: clustering = DBSCAN(eps=0.09, min_samples=9).fit(XTrain)
```

## DBSCAN Implementation Using SKLEARN

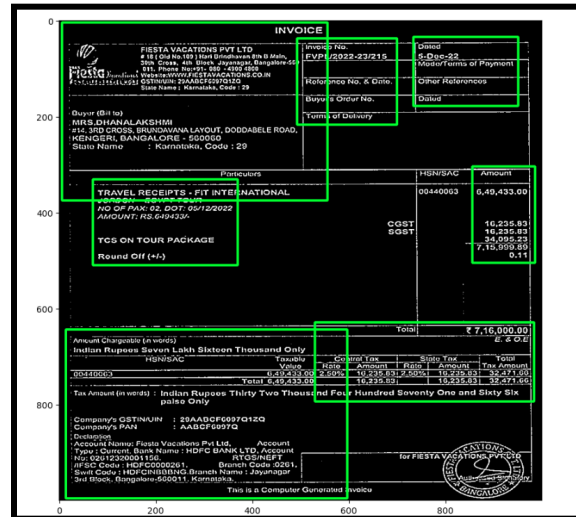
- After this, the DBSCAN clusters were overlapped on the original image to visualize the results.



## DBSCAN Output

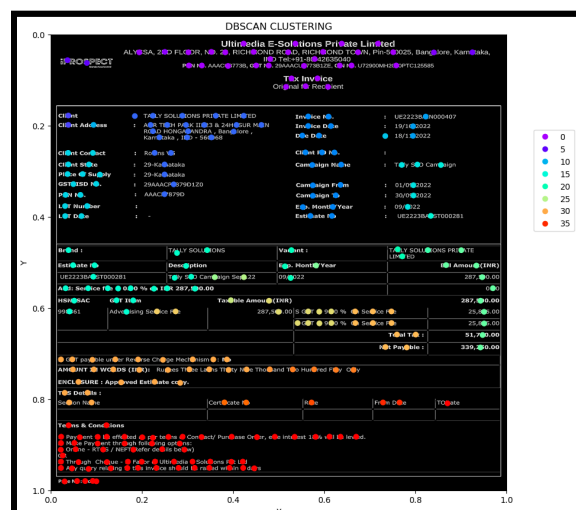
- DBSCAN divided the clusters in a really efficient way, but the same parameters didn't work fine on all images as each image had a different format.

- There was a need for some hyperparameter tuning for DBSCAN to fit each image accurately.
- Since there are only two hyperparameters in DBSCAN, we used a simple nested for loop with a range of 5 points each to tune the parameters for each image separately.
- After applying DBSCAN on 110 images, output was visualized.



DBSCAN Clusters Visualized

- DBSCAN worked fine on many invoices but gave confounding results in some cases and divided key and value in different clusters.



DBSCAN Incorrect Output

- Now, as the output seemed satisfactory for some invoices, this was used to modify the tesseract output and the tokens were concatenated according to the cluster they belong to.
- If the data point belonged to the noise cluster, i.e, -1, it was simply ignored.
- The final dataset after this had 770 rows for 110 images.

|                      | conf      | text   | x      | y      | imageName |
|----------------------|-----------|--|--------|--------|-----------|
| 0                    | 62.857190 | INVOICE FIESTA VAGATIONS PVT LTD Tinvoice ...      | 0.2810 | 0.1895 | 20.jpeg   |
| 1                    | 53.489527 | Tinvoice No.  EVPL/2022-23/215 Reference fio, ...  | 0.5965 | 0.1270 | 20.jpeg   |
| 2                    | 53.916695 | Gated 5-Doc-22 Mod = fodarTarms of Payment 'in...  | 0.8425 | 0.1060 | 20.jpeg   |
| 3                    | 57.832790 | Amount i 6 49,433.00 16,235.83 16,235.83 34 3 ...  | 0.9210 | 0.4045 | 20.jpeg   |
| 4                    | 84.459252 | TRAVEL RECEIPTS FIT YORDON + EGYPT TOUR NO OF ...  | 0.2200 | 0.4205 | 20.jpeg   |
| ...                  | ...       | ...  | ...    | ...    | ...       |
| 765                  | 69.189151 | 28-Sep-22 Mode/Teems of Payment Reference(s) N...  | 0.8425 | 0.1060 | 62.jpeg   |
| 766                  | 60.873739 | Total VATICURY 200.000!                            | 0.9210 | 0.4045 | 62.jpeg   |
| 767                  | 71.821769 | Gescription of Servicios: Markating Exponse (To... | 0.2200 | 0.4205 | 62.jpeg   |
| 768                  | 72.512037 | i Fotat Amount Chargeadie (in words) 'Omani Ri...  | 0.3055 | 0.8200 | 62.jpeg   |
| 769                  | 45.959692 | OMR 200.000! i _ &, OE)                            | 0.7570 | 0.7115 | 62.jpeg   |
| 770 rows x 5 columns |           |  |        |        |           |

Dataset Created By DBSCAN

- Now on analyzing the dataset, we realized that OCR did the job of reading the text but it misspelled most of the words maybe due to the image resizing to 1000px X 1000px.
- We removed the image resize step from the pre-processing and results showed that now the spellings were correct and clustering also improved slightly.
- Now, the final dataset consisted of 1917 rows and we annotated these rows into four classes, namely, invoiceDetails, sellerDetails, buyerDetails and amount.
- After annotation, we had to train a classification model which would predict the class in which each cluster lies.
- We used TFIDF vectorizer to convert the text to trainable features and used chi square similarity to find the most significant keywords for each class.

- 
- Since our dataset was limited to our organization, rather than considering keywords like buyer and seller, it was taking “Tally”, “Solutions” as significant keywords.

```
Class---> amount:
  Most Correlated Unigrams are: total, amount, 00
  Most Correlated Bigrams are: hsn sac, amount in, in words

Class---> buyerDetails:
  Most Correlated Unigrams are: solutions, buyer, to
  Most Correlated Bigrams are: tally solutions, to tally, bill to

Class---> invoiceDetails:
  Most Correlated Unigrams are: 22, 2023, date
  Most Correlated Bigrams are: pvt ltd, amount in, in words

Class---> sellerDetails:
  Most Correlated Unigrams are: limited, company, india
  Most Correlated Bigrams are: amount in, in words, private limited
```

Most Significant Features For Each Class

## Findings :

- DBSCAN algorithm uses distance to nearest neighbor for clustering and is better if the documents are template based, which in our use case is not the scenario and all the invoices have a different template/structure. Hence the DBSCAN output was not that accurate and keys and values lied in different clusters for some cases.
- Text could not be used for cluster classification as it was overfitting due to limited size and variance in the dataset.



## Method 4 : Google's Representation Learning For Information Extraction From Form-Like Documents

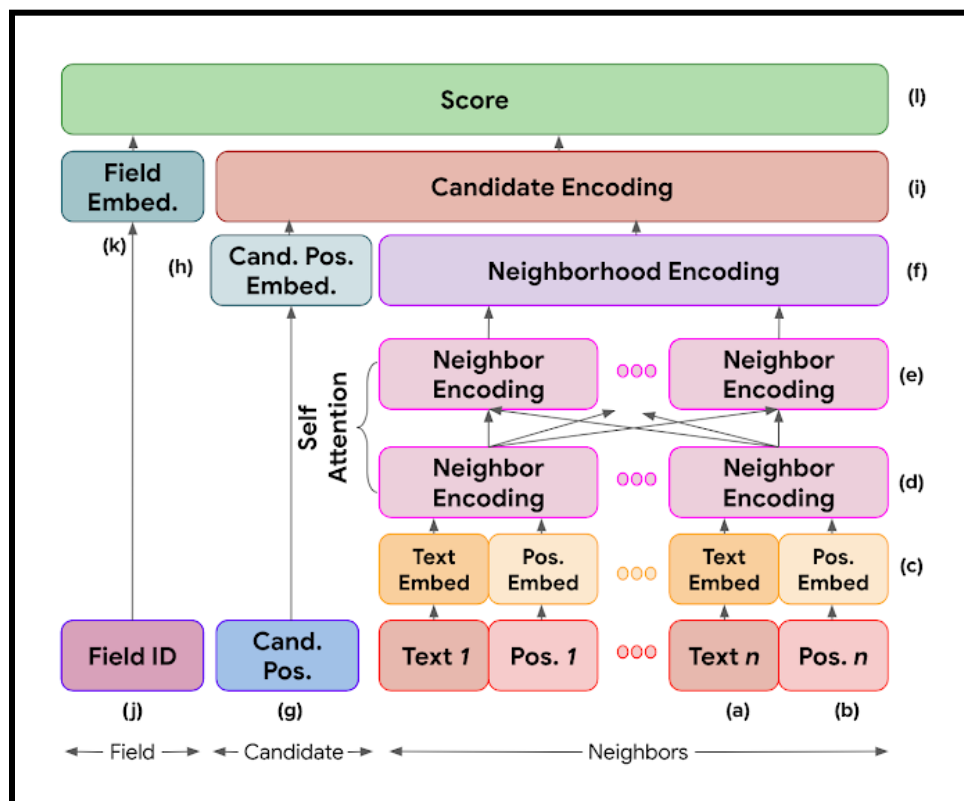
### Implementation

Link to the original research paper :

<https://ai.googleblog.com/2020/06/extracting-structured-data-from.html>

This approach is quite similar to our approach in method 2 mentioned above. There are few changes in encoding techniques and feature selection for model training.

In this, Google used all the candidates for each field, encoded them along with the neighbor information and scored them against the ground truth (true value) encodings of each field.



Model Proposed By Google

## Creating Data :

- For this experiment, first we started with just three fields, namely, InvoiceDate, InvoiceNumber and the TotalAmount.
- For capturing ground truth and creating a dataset, we used RoboFlow tool. We annotated 110 images using this, and exported our dataset in .csv format.

|     | filename | width | height | class         | xmin | ymin | xmax | ymax |
|-----|----------|-------|--------|---------------|------|------|------|------|
| 0   | 68.jpeg  | 2550  | 3300   | InvoiceDate   | 1740 | 540  | 1975 | 625  |
| 1   | 68.jpeg  | 2550  | 3300   | InvoiceNumber | 300  | 860  | 690  | 940  |
| 2   | 68.jpeg  | 2550  | 3300   | TotalAmount   | 1805 | 2405 | 2200 | 2500 |
| 3   | 56.jpeg  | 2550  | 3300   | InvoiceNumber | 168  | 408  | 389  | 455  |
| 4   | 56.jpeg  | 2550  | 3300   | InvoiceDate   | 121  | 470  | 305  | 519  |
| ... | ...      | ...   | ...    | ...           | ...  | ...  | ...  | ...  |
| 316 | 7.jpeg   | 2479  | 3500   | InvoiceNumber | 2025 | 650  | 2288 | 691  |
| 317 | 7.jpeg   | 2479  | 3500   | TotalAmount   | 2087 | 217  | 2284 | 298  |
| 318 | 37.jpeg  | 2550  | 3300   | InvoiceNumber | 1871 | 167  | 2177 | 224  |
| 319 | 37.jpeg  | 2550  | 3300   | InvoiceDate   | 2004 | 233  | 2273 | 278  |
| 320 | 37.jpeg  | 2550  | 3300   | TotalAmount   | 1790 | 2205 | 2010 | 2300 |

321 rows x 8 columns

Annotated Dataset

- For extracting the text from each image, we used TesseractOCR and stored output of each of the 110 images in the form of a .json file.
- Now, we had to extract all the valid candidates for each of the fields in our schema. We used Regex for InvoiceNumber and Amount field and dateparser for the InvoiceDate field.

```
{'invoice.no': [{'text': '441TH',  
'x1': 1646,  
'y1': 163,  
'x2': 1767,  
'y2': 193}],  
{'text': '4', 'x1': 1689, 'y1': 321, 'x2': 1711, 'y2': 351},  
{'text': '4', 'x1': 2145, 'y1': 321, 'x2': 2169, 'y2': 351},  
{'text': '3864542', 'x1': 2184, 'y1': 321, 'x2': 2351, 'y2': 351},  
{'text': '1004849210000003', 'x1': 1131, 'y1': 577, 'x2': 1545, 'y2': 614},  
{'text': 'PCAP/INV/22-23/852', 'x1': 2100, 'y1': 718, 'x2': 2419, 'y2': 749},  
{'text': '1-Mar-23', 'x1': 2101, 'y1': 779, 'x2': 2267, 'y2': 810},  
{'text': 'PQEZC73457877A', 'x1': 2100, 'y1': 840, 'x2': 2421, 'y2': 871},  
{'text': '27-Feb-23', 'x1': 2099, 'y1': 901, 'x2': 2290, 'y2': 932},  
{'text': '100265405000003', 'x1': 296, 'y1': 939, 'x2': 636, 'y2': 970},  
{'text': 'PCAP/ON/22-23/1019', 'x1': 2100, 'y1': 962, 'x2': 2416, 'y2': 993},  
{'text': '7', 'x1': 2099, 'y1': 1024, 'x2': 2119, 'y2': 1053},  
{'text': 'IP13128GB-BLK-N', 'x1': 154, 'y1': 1201, 'x2': 497, 'y2': 1232},  
{'text': '13', 'x1': 719, 'y1': 1202, 'x2': 759, 'y2': 1232},  
{'text': '128GB', 'x1': 776, 'y1': 1201, 'x2': 899, 'y2': 1232},  
{'text': '100', 'x1': 1388, 'y1': 1202, 'x2': 1451, 'y2': 1232},  
{'text': '5', 'x1': 1827, 'y1': 1202, 'x2': 1847, 'y2': 1232},  
{'text': 'IP13128GB-BLU-N', 'x1': 153, 'y1': 1297, 'x2': 499, 'y2': 1328},  
{'text': '13', 'x1': 719, 'y1': 1297, 'x2': 759, 'y2': 1328},  
{'text': '128GB', 'x1': 776, 'y1': 1297, 'x2': 899, 'y2': 1328},  
{'text': '120', 'x1': 1388, 'y1': 1297, 'x2': 1451, 'y2': 1328},
```

Regex Output

- Now, for each candidate we extracted its neighbor information, i.e, neighboring keyword's text, its "X" position and its "Y" position.

```
[[],
[{'text': ['date'], 'X': 0.4183138362242839, 'Y': 0.059931506849315086}],
[{'text': ['date'], 'X': 0.5865268253327955, 'Y': 0.07320205479452055}],
[{'text': ['invoice'], 'X': 0.2045179507866074, 'Y': 0.012842465753424681},
{'text': ['dated'], 'X': 0.018152480839047858, 'Y': 0.012842465753424681}],
[{'text': ['payment'], 'X': 0.03146430012101642, 'Y': 0.007420091324200906}],
[],
[{'text': ['payment'], 'X': 0.11355385235982252, 'Y': 0.09817351598173524},
{'text': ['due'], 'X': 0.14360629286002424, 'Y': 0.06720890410958913},
{'text': ['date'], 'X': 0.03348124243646633, 'Y': 0.06720890410958913},
{'text': ['delivery'],
'X': 0.4945542557482856,
'Y': 0.00042808219178080975}],
[{'text': ['payment'], 'X': 0.2466720451795078, 'Y': 0.09731735159817362},
{'text': ['date'], 'X': 0.032472771278741486, 'Y': 0.09831621004566216},
{'text': ['payment'], 'X': 0.1236385639370714, 'Y': 0.09018264840182655},
{'text': ['due'], 'X': 0.013110125050423416, 'Y': 0.09118150684931514},
{'text': ['due'], 'X': 0.27672448567970953, 'Y': 0.06635273972602751},
{'text': ['date'], 'X': 0.16659943525615162, 'Y': 0.06635273972602751},
{'text': ['payment'], 'X': 0.06171843485276318, 'Y': 0.06521118721461194}],
[{'text': ['due'], 'X': 0.1393707139975796, 'Y': 0.07919520547945214},
{'text': ['date'], 'X': 0.029245663574021696, 'Y': 0.07919520547945214},
{'text': ['delivery'], 'X': 0.49031867688584096, 'Y': 0.012414383561643816}],
[],
[{'text': ['delivery'], 'X': 0.45018152480839047, 'Y': 0.03652968036529669}],
[{'text': ['delivery'], 'X': 0.4497781363453005, 'Y': 0.048373287671232834}],
[]]
```

Output Of Neighbor Extraction

- Now we created a word embedding table to encode each keyword into a meaningful numerical value. This was done by assigning the same value to synonyms and similar sounding words.

```
vocab = ["invoice", "inv", "receipt", "bill", "order", "payment", "due", "date", "dated"]
emb = {
    "invoice" : 1,
    "inv" : 1,
    "bill" : 1,
    "receipt" : 1,
    "order" : 2,
    "payment" : 3,
    "due" : 4,
    "date" : 5,
    "dated" : 5
}
```

Word Embedding Table

- Now, we used PCA (Principal Component Analysis) to encode these three variables into a single variable for training.

```
[58] from sklearn.decomposition import PCA
      ✓ 0.0s

[59] pca = PCA(n_components=1)
      ✓ 0.0s

[60] res = pca.fit_transform(xTrain)
      ✓ 0.0s
```

PCA

- We trained similar PCA models for candidate positions also and repeated the steps for all of the 110 images.
- Our final data looked like this.

|    | Unnamed: 0 | text        | CandidatePosition | NeighbourInformation |
|----|------------|-------------|-------------------|----------------------|
| 0  | 0          | 31-Dec-22   | 0.355992          | 0.269873             |
| 1  | 1          | 9902016147  | 0.369481          | 0.269891             |
| 2  | 2          | Invoice     | 0.138290          | 0.269530             |
| 3  | 3          | Desk        | -0.244673         | -0.493927            |
| 4  | 4          | INTEGRATED  | 0.222155          | -0.493927            |
| 5  | 5          | +91         | -0.124758         | -0.493927            |
| 6  | 6          | 6661        | -0.022596         | -0.493927            |
| 7  | 7          | 7000        | 0.048244          | -0.493927            |
| 8  | 8          | 18%         | 0.449655          | -0.493927            |
| 9  | 9          | 18%         | 0.449655          | -0.493927            |
| 10 | 10         | _76,888.40  | 0.680470          | -0.493927            |
| 11 | 11         | 4,26,880.00 | 0.388924          | -0.493927            |
| 12 | 12         | 76,838.40   | 0.686578          | -0.493927            |
| 13 | 13         | DEC.'22     | -0.444526         | -0.493927            |
| 14 | 14         | Desk        | -0.369990         | -0.493927            |
| 15 | 15         | 2%          | -0.408592         | -0.493927            |
| 16 | 16         | 6 7         | 0.614823          | -0.493927            |

Dataset

---

## Annotating Data :

- We used our RoboFlow annotations to annotate this dataset as a binary classifier with true invoice dates as 1 and other candidates as 0.

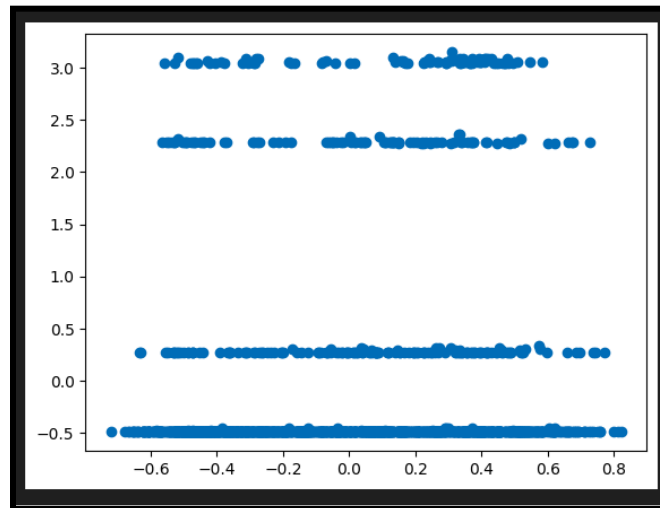
|    | text          | CandidatePosition | NeighbourInformation | Output |
|----|---------------|-------------------|----------------------|--------|
| 0  | \$22          | -0.641336         | -0.493927            | 0      |
| 1  | 04/11/2022    | -0.631070         | 0.269386             | 1      |
| 2  | 1499          | -0.511586         | 0.269756             | 0      |
| 3  | 9.00 1499.00  | -0.363099         | 0.270348             | 0      |
| 4  | 1499.00 MENS  | -0.503532         | -0.493927            | 0      |
| 5  | 1499          | -0.501412         | -0.493927            | 0      |
| 6  | 9.00 1499.00  | -0.352926         | -0.493927            | 0      |
| 7  | 1499.00 MENS  | -0.493359         | -0.493927            | 0      |
| 8  | 8901326000311 | -0.606000         | -0.493927            | 0      |
| 9  | 2.50 240.90   | -0.344970         | -0.493927            | 0      |
| 10 | 1499 9.00     | -0.454457         | -0.493927            | 0      |
| 11 | 9.00 1499.00  | -0.332579         | -0.493927            | 0      |
| 12 | 1499.00 5     | -0.469936         | -0.493927            | 0      |
| 13 | 5 201250239   | -0.607681         | -0.493927            | 0      |
| 14 | 9.00 899.00   | -0.324669         | -0.493927            | 0      |
| 15 | 4807.90       | -0.306172         | -0.493927            | 0      |
| 16 | 5636.90       | -0.270322         | 0.270270             | 0      |
| 17 | 5636.90       | -0.251935         | -0.493927            | 0      |

Dataset After Annotation

- Now each of the images had one true value among all the candidates for each field.
- For experimental purposes we just started with a single field, i.e, invoiceDate field.

## Processing Data :

- Now, on plotting the two input variables, i.e, "CandidatePosition" and "NeighbourInformation", we found out that NeighbourInformation is broadly classified into 4 main categories.



Scatter Plot Of Input Variables

- So we used clustering to convert the NeighbouringInformation column into a categorical column (0, 1, 2, 3) and then used the get\_dummies function to convert it into four columns.

|      | CandidatePosition | NeighbourInformation | Output | NeighbourClass_0 | NeighbourClass_1 | NeighbourClass_2 | NeighbourClass_3 |
|------|-------------------|----------------------|--------|------------------|------------------|------------------|------------------|
| 0    | -0.486797         | -0.475342            | 0      | True             | False            | False            | False            |
| 1    | -0.473302         | -0.475340            | 0      | True             | False            | False            | False            |
| 2    | 0.575232          | 0.330898             | 1      | False            | True             | False            | False            |
| 3    | -0.535680         | -0.493927            | 0      | True             | False            | False            | False            |
| 4    | 0.651722          | -0.493927            | 0      | True             | False            | False            | False            |
| ...  | ...               | ...                  | ...    | ...              | ...              | ...              | ...              |
| 1410 | 0.606355          | -0.493927            | 0      | True             | False            | False            | False            |
| 1411 | 0.605394          | -0.457813            | 0      | True             | False            | False            | False            |
| 1412 | -0.467087         | -0.493923            | 0      | True             | False            | False            | False            |
| 1413 | -0.261925         | 0.269141             | 0      | False            | True             | False            | False            |
| 1414 | 0.032232          | -0.493927            | 0      | True             | False            | False            | False            |

1415 rows x 7 columns

Final Dataset For Training

---

## Approach 1 :

- Now we trained a XGBoost classifier on this dataset.
- The best model achieved using GridSearchCV is shown below.

```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
               colsample_bylevel=None, colsample_bynode=None,
               colsample_bytree=None, early_stopping_rounds=None,
               enable_categorical=False, eval_metric='error', feature_types=None,
               gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
               interaction_constraints=None, learning_rate=0.08, max_bin=None,
               max_cat_threshold=None, max_cat_to_onehot=None,
               max_delta_step=None, max_depth=6, max_leaves=None,
               min_child_weight=None, missing=nan, monotone_constraints=None,
               n_estimators=10, n_jobs=None, num_parallel_tree=6,
               objective='binary:hinge', predictor=None, ...)
```

XGBoost model used

- Using this we achieved an accuracy of 89%.

```
Total : 283

True +ve :   244      93.85 %
True -ve :    8      34.78 %
False +ve :   16       6.15 %
False -ve :   15      65.22 %

Accuracy :   89.05 %
```

Model Metrics

- But since our data was highly imbalanced in nature, accuracy can not be used as a perfect metric to evaluate this model. Our model was able to predict only 8 out of 23 images correctly.

## Approach 2 :

- Now, we changed hard encodings for neighbor keywords, which were confined to just ~10 keywords to encodings created using a word embedding model and which was trained on the whole corpus rather than just some predefined keywords.
- The final json for all the keywords and encodings was having 3551 unique keywords.

```
{'invoice': -0.03879399225115776,
'pitstop': -0.005783237516880035,
'order': 0.031136836856603622,
'id': 0.010334979742765427,
'service': 0.0252956785261631,
'date': -0.03593393415212631,
'delivery': -0.021567273885011673,
'registration': 0.008700598031282425,
'number': 0.03502375259954605,
'renault': -0.04446839168667793,
'pulse': -0.019214725121855736,
'diesel': -0.04597220569849014,
'odometer': 0.02414466068148613,
'from': 0.03355500474572182,
'to': 0.03222217783331871,
'cars': 0.03802556172013283,
'care': -0.02098955027759075,
'opp': -0.007129956036806107,
'trident': 0.01780029758810997,
'hyundai': 0.03034130111336708,
'showroom': -0.028101205825805664,
'kudlu': 0.02260586991906166,
'gate': -0.026707543060183525,
'hosur': 0.03071046993136406,
'main': -0.049713920801877975,
...
'daciaration': -0.0063305869698524475,
'nb': -0.003580927848815918,
'barang': 0.03836692497134209,
'yang': -0.0007471665740013123,
...}
```

### Neighbor Keyword Embeddings JSON File

- Now neighbor along with neighbor's relative position was assigned to each of the candidates.

|   | text                   | x1       | y1       | x2       | y2       | Class        | invoice   | invoice_X | invoice_Y | pitstop | ... | tunga_Y | chambers | chambers_X | chambers_Y | kh | kh_X | kh_Y | kkbkinb | kkbki |
|---|------------------------|----------|----------|----------|----------|--------------|-----------|-----------|-----------|---------|-----|---------|----------|------------|------------|----|------|------|---------|-------|
| 0 | Bhavani                | 0.130698 | 0.081050 | 0.188786 | 0.089897 | invoice_date | 0.000000  | 0.000000  | 0.0000    | 0       | ... | 0       | 0        | 0          | 0          | 0  | 0    | 0    | 0       | 0     |
| 1 | 21-07-2022             | 0.692618 | 0.073345 | 0.772489 | 0.080765 | invoice_date | -0.038794 | 0.223881  | 0.0127    | 0       | ... | 0       | 0        | 0          | 0          | 0  | 0    | 0    | 0       | 0     |
| 2 | 24,                    | 0.300121 | 0.197203 | 0.323114 | 0.208333 | invoice_date | 0.000000  | 0.000000  | 0.0000    | 0       | ... | 0       | 0        | 0          | 0          | 0  | 0    | 0    | 0       | 0     |
| 3 | 1250                   | 0.590561 | 0.352740 | 0.626059 | 0.361301 | invoice_date | 0.000000  | 0.000000  | 0.0000    | 0       | ... | 0       | 0        | 0          | 0          | 0  | 0    | 0    | 0       | 0     |
| 4 | 9 CGST 9%              | 0.274708 | 0.447774 | 0.649455 | 0.475457 | invoice_date | 0.000000  | 0.000000  | 0.0000    | 0       | ... | 0       | 0        | 0          | 0          | 0  | 0    | 0    | 0       | 0     |
| 5 | 32,812.00<br>31,250.00 | 0.350545 | 0.618721 | 0.901573 | 0.638699 | invoice_date | 0.000000  | 0.000000  | 0.0000    | 0       | ... | 0       | 0        | 0          | 0          | 0  | 0    | 0    | 0       | 0     |
| 6 | 32,812.00<br>Company's | 0.061718 | 0.643550 | 0.901573 | 0.684075 | invoice_date | 0.000000  | 0.000000  | 0.0000    | 0       | ... | 0       | 0        | 0          | 0          | 0  | 0    | 0    | 0       | 0     |
| 7 | 1168120020000193       | 0.127874 | 0.716895 | 0.288826 | 0.726027 | invoice_date | 0.000000  | 0.000000  | 0.0000    | 0       | ... | 0       | 0        | 0          | 0          | 0  | 0    | 0    | 0       | 0     |
| 8 | Bhavani                | 0.603873 | 0.859018 | 0.672449 | 0.872146 | invoice_date | 0.000000  | 0.000000  | 0.0000    | 0       | ... | 0       | 0        | 0          | 0          | 0  | 0    | 0    | 0       | 0     |

9 rows x 10659 columns

### Neighbor Assignment



- Now this dataset was processed and neighbor fields having zero significance to the output were dropped from the dataset.
- The final dataset now had 5723 columns for each image.
- Columns were broadly divided into :
  - Candidate position : 4 columns
  - Neighbor Keywords : 1906 columns
  - Neighbor X positions : 1906 columns
  - Neighbor Y positions : 1906 columns
  - Output : 1 column
- This dataset was highly imbalanced and had just a few hundred positives and a thousand of negatives.
- We used oversampling to make this data balanced by adding synthetic positive data points.
- Now a keras ANN model was trained on this dataset.

```
Model: "sequential_1"
```

| Layer (type)     | Output Shape | Param #  |
|------------------|--------------|----------|
| dense_3 (Dense)  | (None, 2048) | 11720704 |
| dense_4 (Dense)  | (None, 2048) | 4196352  |
| dense_5 (Dense)  | (None, 1024) | 2098176  |
| dense_6 (Dense)  | (None, 512)  | 524800   |
| dense_7 (Dense)  | (None, 256)  | 131328   |
| dense_8 (Dense)  | (None, 128)  | 32896    |
| dense_9 (Dense)  | (None, 64)   | 8256     |
| dense_10 (Dense) | (None, 32)   | 2080     |
| dense_11 (Dense) | (None, 1)    | 33       |

```

Total params: 18,714,625
Trainable params: 18,714,625
Non-trainable params: 0

```

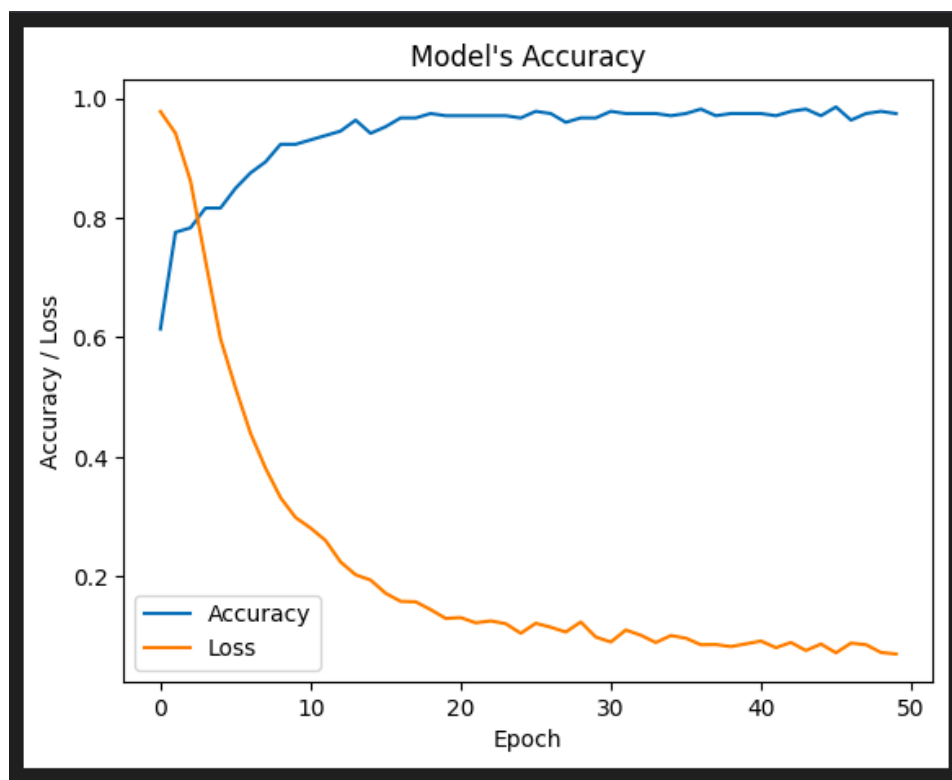
Keras model summary

- This model was now trained for 100 epochs and a batch size of 32 and final training precision score was 72.97% with an accuracy of 96.73%.
- This model didn't work well on the test dataset, mainly due to the overfitting arised by oversampling to a great extent.

---

### Approach 3 :

- Now since oversampling was leading to overfitting, we needed a mixture of undersampling and oversampling to conquer this imbalance in the dataset.
- For this we changed and refined our candidate extraction techniques significantly and managed to constraint each image to have about 4-5 candidates at maximum.
- Using this technique, we were able to reduce our negative points in the dataset from about 2000 to 300. This improved the imbalance significantly.
- Now rather than oversampling, we used class weights to take care of the imbalance and gave three times more weight to the positive class.
- We now fitted a keras model similar to the above model and with little hyperparameter tweaking, we achieved a F1 score of 0.62.



Model Training On 50 Epochs

- Now we tried fitting a XGBoost model on this dataset and after hyperparameter tuning using grid search we were able to achieve a maximum F1 score on 0.86

```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
               colsample_bylevel=None, colsample_bynode=None,
               colsample_bytree=None, early_stopping_rounds=None,
               enable_categorical=False, eval_metric='error', feature_types=None,
               gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
               interaction_constraints=None, learning_rate=0.09, max_bin=None,
               max_cat_threshold=None, max_cat_to_onehot=None,
               max_delta_step=None, max_depth=10, max_leaves=None,
               min_child_weight=None, missing=nan, monotone_constraints=None,
               n_estimators=500, n_jobs=None, num_parallel_tree=7,
               objective='binary:hinge', predictor=None, ...)
```

Tuned XgBoost Model

- Now, the output was satisfactory but still our dataset had an imbalance which was taken care of on Keras model but not in the XgBoost model.
- To remove this imbalance completely, we now applied oversampling on our dataset and since now the ratio of majority class to minority class was not that high, it did not lead to overfitting on the training set.
- Now on fitting this oversampled dataset on our XgBoost model, we were able to achieve a F1 score as high as 0.96 on the unseen test dataset, with 19/20 images predicted correctly.

```
Total : 69

True +ve : 46      93.88 %
True -ve : 19      95.00 %
False +ve : 3       6.12 %
False -ve : 1       5.00 %

Accuracy : 94.20 %

F1 Score : 0.96
```

Test Results on XgBoost Model

## Model Testing :

- Now for the model inference we created a function that takes an image as an input, pre-processes it so as to fit our model and applies the model on that image to return the output.
- Model's predictions were then visualized using matplotlib.

CION LIGHTING TECHNOLOGIES FZC  
TRN: 100549453700003

**TAX INVOICE**

Bill To :  
Name : ANTIC-CUSTOMER  
Address : Al Saadon Building 2nd floor, Salahuddeen Ayyoubi Road, Mutez, P.O Box 65215 Riyadh - KSA, Tel: +966-11-4773321  
City : Riyadh  
Country : Saudi Arabia  
Phone :  
Email :  
Cust. TRN : 300052062600003

Invoice No.: CIL-47  
Invoice Date: 16-3-2023  
Sales Order No.: S003220596  
LPO: S003220596  
Del. Note No.: 15-March-23  
Del. Note Date: 16-3-2023  
Project Number: R&B FORECAST, DEERAH OLD, CP @ PANORAMA, ANTIC STK-RSA  
Place of Supply: KSA  
Page No.: 1

| S.No. | Item             | Description   | Qty        | U. Rate | Amount (AED) | VAT Rate | VAT Amount |
|-------|------------------|---|------------|---------|--------------|----------|------------|
| 1     | ADA-GAB9AD3-B    | GAB9AD3-B 3 PHASE TRACK ADAPTER BLACK                                       | 20.000 NCS | 12.90   | 2,579.50     | 0 %      |            |
| 2     | CBC-GAB9HOLDR    | CONNECTOR CABLE HOLDER FOR GAB9 ADAPTER                                     | 20.000 NCS | 0.74    | 147.40       | 0 %      |            |
| 3     | DGL-CXG2BL30N-R  | CION CXG2BL30N-R LLA 3000K LED PREMIUM DOUBLE GIMBAL WITH WALL WASHER WHITE | 16.000 NCS | 276.38  | 49,748.00    | 0 %      |            |
| 4     | SGL-CXG2BL30N-R  | CION CXG2BL30N-R LLA 3000K LED PREMIUM SINGLE GIMBAL WITH WALL WASHER WHITE | 08.000 NCS | 141.87  | 89,380.00    | 0 %      |            |
| 5     | DRV-TRL04-FIX2W  | TRIDONIC LC44W 87500960 1050mA/42 FwC SRL ADV2 LED DRIVER WITH WIRE         | 07.000 NCS |         |              | 0 %      |            |
| 6     | TRL-CTF51020W-8W | CION CXTF51-SM LDA 3000K LED TRACK LIGHT WHITE                              | 07.000 NCS | 138.19  | 7,876.70     | 0 %      |            |
| 7     | DRV-TRL04-FIXC   | TRIDONIC LC44W 87500960 1050mA/42 FwC SRL ADV2 LED DRIVER                   | 07.000 NCS |         |              | 0 %      |            |
| 8     | TRL-CTF51020W-R2 | CION CXTF51-R LDA 3000K/32W LED TRACK LIGHT WHITE & DRIVER                  | 10.000 NCS | 138.19  | 104,055.00   | 0 %      |            |
| 9     | SSP-PRO-0348-3-W | PRO-0348-3-W SINGLE PHASE TRACK SUSPENSION KIT 3M CHAN WHITE PJ3            | 08.000 NCS | 11.06   | 6,636.00     | 0 %      |            |
| 10    | SSP-PRO-0348-3-W | PRO-0348-3-W SINGLE PHASE TRACK SUSPENSION KIT 3M CHAN WHITE PJ3            | 08.000 NCS | 11.06   | 2,212.00     | 0 %      |            |

Continue .....

CION LIGHTING TECHNOLOGIES FZC  
H2-05, SAF ZONE, P.O Box 120589 | Sharjah - UAE  
TRN: 100549453700003

Tel: +971 6 5440833, PO Box-120589  
Email: finance@cionlighting.com  
www.cionlighting.com

Model's Prediction

- Here, the red boxes show all the candidates for the Invoice Date field and the green box shows the prediction made by the model.
- This model took around 10 seconds on an average to predict and visualize the output of one image.

---

## Final Model : Based on Google's Research Paper

### Introduction :

- This model is based on the learnings from Google's research paper on representation learning.
- This model infuses the ideas proposed by Google in our model defined in method 2 of this paper.
- This model is similar to Google's model in various aspects but differ in :
  - Feature Extraction, i.e, number of features extracted. Google decomposed all the features into just two features but we here did not do that so as to not compromise with the accuracy, since the dataset we used was already very limited.
  - Final model trained. Google trained a single model for all the fields but we here have trained multiple models, one for each field, as binary classifiers.
- We here have trained models for extracting three major fields as of now, namely,
  - Invoice Date
  - Invoice Number
  - Total Amount
- Major technologies and python libraries used :
  - Pandas : Working with dataframes
  - Tesseract : OCR Engine
  - Tensorflow : Framework for training ML models
  - Keras : Library based on tensorflow
  - Scikit Learn : Utilities like scoring and confusion matrices
  - XgBoost : Model used for training
  - Matplotlib : Visualization
  - Regex : Regular Expressions

---

## Workflow :

1. Applying **Tesseract OCR** to extract all the text from documents.
2. **Extracting candidates** for each field in the schema.
3. **Keras word embedding model** for encoding neighbor text.
4. Assigning **neighbor information** to each of the candidates in the dataset.
5. **Pre-processing the dataset** in order to select valuable neighbor fields.
6. **Training and testing** multiple models for multiple fields in the schema.
7. **Inference** of the model on new and unseen documents.
8. **Benchmark** the performance and accuracy of the model.

## Tesseract OCR :

- For this model, we decided use tesseract OCR engine with the following configuration :
  - "--oem1 --psm3"
    - OEM 1 refers to the "**Legacy OCR Engine**" mode, which was the original OCR engine used in Tesseract.
    - PSM 3 corresponds to "**Automatic page segmentation with OSD**" (Orientation and Script Detection).
- Now, the tesseract output was processed to transform some columns and centroid information was added for each bounding box.

```
def ocr(imagePath):  
    image = cv2.imread(imagePath)  
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  
    result = pytesseract.image_to_data(image, config='--oem 1 --psm 3', output_type=Output.DATFRAME)  
    df = pd.DataFrame(result)  
    df.dropna(inplace=True)  
    df["x1"] = df["left"]  
    df["y1"] = df["top"]  
    df["x2"] = df["left"] + df["width"]  
    df["y2"] = df["top"] + df["height"]  
    df["X"] = (df["x1"] + df["x2"])/2  
    df["Y"] = (df["y1"] + df["y2"])/2  
    return df
```

Tesseract OCR Function

---

## Candidate Extraction :

- For each document in the dataset, OCR described above was applied on it and a dataframe was created.
- From that dataframe, candidates for each of the fields in our schema were extracted using various techniques.
  - **Invoice Date** : Dateparser module was used here to extract all the date fields.
  - **Invoice Number** : Simple RegEx code was written to extract this field.
  - **Total Amount** : Simple RegEx code was written to extract this field.
- Now the output from this extraction was further modified so as to reduce the number of candidates extracted per field. Main idea behind this was to reduce the imbalance between true and false outputs.

|   | text       | x1       | y1       | x2       | y2       | Class        |
|---|------------|----------|----------|----------|----------|--------------|
| 0 | 8th        | 0.356436 | 0.074286 | 0.371089 | 0.081143 | invoice_date |
| 1 | 5-Dec-22   | 0.624554 | 0.073143 | 0.689109 | 0.081714 | invoice_date |
| 2 | 30th       | 0.177030 | 0.084857 | 0.197228 | 0.091714 | invoice_date |
| 3 | 4th        | 0.243960 | 0.084857 | 0.258614 | 0.091429 | invoice_date |
| 4 | 05/12/2022 | 0.253465 | 0.326857 | 0.323960 | 0.334571 | invoice_date |
| 5 | 3rd        | 0.070495 | 0.778286 | 0.088713 | 0.788286 | invoice_date |

Extracted Candidates For InvoiceDate Field

---

## Word Embedding Model :

- Whole corpus of 110 images were trained on a keras embedding model so as to give a single number for each word(token) in our documents.
- Corpus consisted of a total of 3551 tokens.

```
word_input = Input(shape=(maxlen, ), dtype="float64")
word_embeddings = Embedding(input_dim=vocab_size, output_dim=OD, input_length=maxlen)(word_input)
word_vec = Flatten()(word_embeddings)
embed_model = Model([word_input], word_vec)
```

### Model Used

- Now this model was used to predict encodings for tokens in our corpus and they were stored in a json file.

```
{'invoice': -0.03879399225115776,
 'pitstop': -0.005783237516880035,
 'order': 0.031136836856603622,
 'id': 0.010334979742765427,
 'service': 0.0252956785261631,
 'date': -0.03593393415212631,
 'delivery': -0.021567273885011673,
 'registration': 0.008700598031282425,
 'number': 0.035023752599954605,
 'renault': -0.04446839168667793,
 'pulse': -0.019214725121855736,
 'diesel': -0.04597220569849014,
 'odometer': 0.02414466068148613,
```

### Model's Results



## Neighbor Extraction :

- We added 3 columns for each token in our corpus in our dataset,
  - Column 1 : Neighbour Text
  - Column 2 : X position
  - Column 3 : Y position
- So now we added  $3551 * 3$ , i.e, 10653 more columns in our dataset.
- Now these columns were assigned values according to their presence near the candidate, i.e, top and left of the candidate.
- Same steps were repeated for datasets of all the three fields in our schema.

|   | text       | x1       | y1       | x2       | y2       | Class        | Invoice   | Invoice_X | Invoice_Y | pltstop | ... | tunga_Y | chambers | chambers_X | chambers_Y |
|---|------------|----------|----------|----------|----------|--------------|-----------|-----------|-----------|---------|-----|---------|----------|------------|------------|
| 0 | 8th        | 0.356436 | 0.074286 | 0.371089 | 0.081143 | invoice_date | 0.000000  | 0.000000  | 0.000000  | 0       | ... | 0       | 0        | 0          | 0          |
| 1 | 5-Dec-22   | 0.624554 | 0.073143 | 0.689109 | 0.081714 | invoice_date | -0.038794 | 0.193267  | 0.014143  | 0       | ... | 0       | 0        | 0          | 0          |
| 2 | 30th       | 0.177030 | 0.084857 | 0.197228 | 0.091714 | invoice_date | 0.000000  | 0.000000  | 0.000000  | 0       | ... | 0       | 0        | 0          | 0          |
| 3 | 4th        | 0.243960 | 0.084857 | 0.258614 | 0.091429 | invoice_date | 0.000000  | 0.000000  | 0.000000  | 0       | ... | 0       | 0        | 0          | 0          |
| 4 | 05/12/2022 | 0.253465 | 0.326857 | 0.323960 | 0.334571 | invoice_date | 0.000000  | 0.000000  | 0.000000  | 0       | ... | 0       | 0        | 0          | 0          |
| 5 | 3rd        | 0.070495 | 0.778286 | 0.088713 | 0.788286 | invoice_date | 0.000000  | 0.000000  | 0.000000  | 0       | ... | 0       | 0        | 0          | 0          |

6 rows × 16 columns

Sample Output For One Document

## Pre-Processing Dataset :

- Firstly the dataset was annotated and a column namely, "Output" was added so as to define whether the candidate is a true value or not.
- For annotation we used a software named Roboflow, which is an online tool and returns a csv file for each image.
- After annotation all the three datasets were filtered separately so as to reduce the number of neighbor features they had.
- Neighbor features having the same value in all images, were simply dropped.
- Now valid keywords used for each dataset, i.e, each field were stored in json files.

```
ID : 799
TA : 555
IN : 1172
```

Neighbor Keywords Per Field

---

## Training and Testing :

- Dataset for each of the fields was split into training and testing sets.
- The training datasets were **oversampled** so as to remove the imbalance completely, and this did not lead to overfitting as we have already reduced the number of negative candidates to a great extent.
- **XgBoost classifier** was used to train all the three models separately on their respective datasets.
- XgBoost models were tuned using **RandomizedSearchCV**.

| MODEL                | F1 SCORE |
|----------------------|----------|
| Invoice Date Model   | 0.96     |
| Invoice Number Model | 0.91     |
| Total Amount Model   | 0.84     |

F1 Scores For Each Model

|                            |                            |
|----------------------------|----------------------------|
| Total : 69                 | Total : 85                 |
| True +ve : 46      93.88 % | True +ve : 60      93.75 % |
| True -ve : 19      95.00 % | True -ve : 13      61.90 % |
| False +ve : 3      6.12 %  | False +ve : 4      6.25 %  |
| False -ve : 1      5.00 %  | False -ve : 8      38.10 % |
| Accuracy : 94.20 %         | Accuracy : 85.88 %         |
| F1 Score : 0.96            | F1 Score : 0.91            |

Invoice Date

Invoice Number

---

```
Total : 37

True +ve : 24      92.31 %
True -ve : 4       36.36 %
False +ve : 2       7.69 %
False -ve : 7       63.64 %

Accuracy : 75.68 %

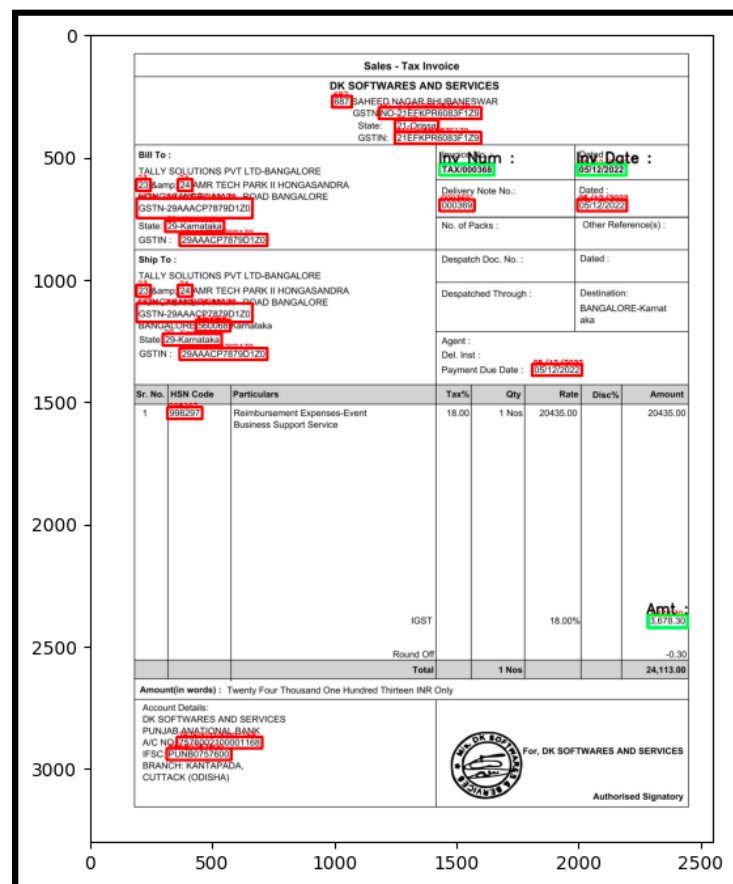
F1 Score : 0.84
```

Total Amount

- After training and testing, XgBoost models were saved as json files.

## Inference :

- Inference and visualization was done using openCV and results were very accurate for the invoice date field, but not that accurate for other two fields, despite the fact that F1 scores for all three fields looked good.
- For inference, the image had to be processed three times to fit three different models. These steps were performed serially.



### Inference Result For One Image

- Here, red boxes define all the candidates and green boxes define the candidate chosen by the model to be the true output.

---

## Benchmark :

- System on which models were trained and tested had following specifications.
  - OS - Ubuntu 22.04 64-bit
  - CPU - Intel CORE i5 10th Gen 8 Cores
  - GPU - NVIDIA GTX 1650 Ti 4 GB
  - RAM - 16 GB
- CPU and Memory usage and time taken for pre-processing the data to fit the models is shown below.

```
BENCHMARK FOR 3 FIELDS  
  
AVERAGE TIME TAKEN : 28.855327129364014  
MAX CPU USED: 19.740000000000002 %  
MAX MEMORY USED: 766.0 MB
```

Pre-Processing For Three Fields

- It took around 35 seconds to predict all the three fields for each image.

---

## **Limitations :**

- Since fields like Invoice Number do not have a format and they can be any numeric/alphanumeric value, candidate extraction for these fields didn't work great and lead to a large imbalance in the final dataset
- Neighborhood vocabulary for all the fields in the schema was different and of variable size.
- Words like "Tally" were also a part of neighborhood but these actually do not define the fields.

## **Future Improvements :**

- Models can be run parallel rather than in a serial manner so as to improve the prediction time.
- For data processing, "Polars" library can be used rather than "Pandas". Polars is similar to Pandas but since Polars is written in "Rust", it is very fast as compared to pandas.
- Candidate extraction techniques for a few fields need to be improved.
- Zonal information can be added to the data so as to improve results for fields like invoice number and amount

---