<u>DOCUMENT-AI</u>
<u>Tally Solutions Pvt. Ltd.</u>

Approach 2 :
Pytesseract – wrapper for the Google Tesseract Engine

<u>Dataset creation from documents :</u>

Step 1 : Document Pre Processing

- Documents of all format were converted to a uniform "*.jpeg" image format using python's pdf2img library.

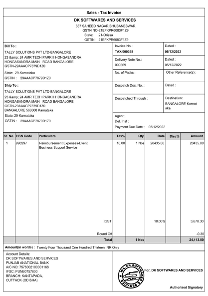
```
In [1]: import os
        import pdf2image
        from pdf2image import convert_from_path
        import cv2
        import matplotlib.pyplot as plt
        import PyPDF2
        import PIL
        from PIL import Image, ImageTk
```

<u>Libraries</u>

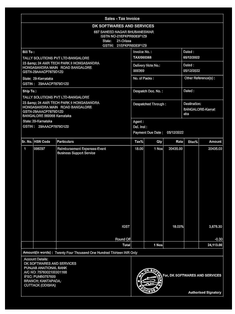
- Then each image was pre processed for achieving better accuracy in the OCR(Optical Character Recognition) task using cv2 library.
  - Grayscale Conversion : Each image in the dataset was converted to grayscale.
  - Gaussian Blur : Gaussian blur was applied to each image of the dataset.
  - Thresholding : Binary thresholding was done on each image of the dataset for increasing the contrast of the image.

```
In [2]: def preProcessImage(imagePath):
            image = cv2.imread(imagePath)
            gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
            plt.imshow(gray, cmap="gray")
            plt.show()
            blur = cv2.GaussianBlur(gray, (3,3),1)
            plt.imshow(blur, cmap="gray")
            plt.show()
            threshold_img = cv2.adaptiveThreshold(blur,255,1,1,11,2)
            plt.imshow(threshold_img, cmap="gray")
            plt.show()
            return threshold_img
```

<u>Image pre processor</u>



<u>Before And After Pre Processing</u>
---------------------->



Step 2 : Applying OCR to extract a DataFrame from image

- Python’s pytesseract wrapper was used for this purpose.

```
In [107]: D = pytesseract.image_to_data(image, output_type=Output.)
                                                       BYTES
                                                       DATAFRAME
                                                       DICT
                                                       mro
                                                       STRING
In [108]: D.keys()
Out[108]: dict_keys(['level', 'page_num', 'block_num', 'par_num', 'word_num', 'left', 'top', 'width', 'height',
'conf', 'text'])
```

Pytesseract

  - Pytesseract has options to get the output in various formats like dictionary, dataframe, string, etc.

- Output was retreived in form of a pytesseract dataframe and was then converted to the pandas datframe.

```
In [72]: df = pd.DataFrame(pytesseract.image_to_data(image, output_type=Output.DATAFRAME))

In [73]: df
Out[73]:
```

	level	page_num	block_num	par_num	line_num	word_num	left	top	width	height	conf	text
0	1	1	0	0	0	0	0	0	1700	2200	-1.000000	NaN
1	2	1	1	0	0	0	654	125	460	68	-1.000000	NaN
2	3	1	1	1	0	0	654	125	460	68	-1.000000	NaN
3	4	1	1	1	1	0	654	125	460	26	-1.000000	NaN
4	5	1	1	1	1	1	654	125	44	26	90.456886	OK)
...	...	...	...	...	...	...	...	...	...	...	...	...
131	5	1	9	1	4	2	271	2008	114	38	65.808868	(ODISHA)
132	2	1	10	0	0	0	111	43	1534	2071	-1.000000	NaN
133	3	1	10	1	0	0	111	43	1534	2071	-1.000000	NaN
134	4	1	10	1	1	0	111	43	1534	2071	-1.000000	NaN
135	5	1	10	1	1	1	111	43	1534	2071	95.000000	

136 rows × 12 columns

Retreived DataFrame

- It resulted in a DataFrame having 12 columns.
  - Level
  - page_num
  - block_num
  - par_num
  - line_num
  - word_num
  - left
  - top
  - width
  - height
  - conf
  - text

Step 3 : Pre Process the dataframe

- It included,
  - removing null values
  - droping irelevent columns
  - removing stopwords
  - adding new box centroid columns using bounding boxes

```
In [4]: def preProcessDataFrame(df):
            #drop null values
            df.dropna(inplace=True)

            #columns having same values throughout removed
            toDrop = ["level", "page_num"]
            df.drop(columns=toDrop, inplace=True)

            #rows having text as a stopword removed
            indexesToDrop = []
            stopwords = ["", " ", "  ", "/", ":", "-", ".", ",", "\n", "\t", "\\", "(", ")", "[", "]", "{", "}", "*", "&", '
            for index in df.index:
                if df["conf"][index]<=40:
                    indexesToDrop.append(index)
                elif df["text"][index] in stopwords:
                    indexesToDrop.append(index)
            df.drop(indexesToDrop, inplace=True)

            #resetting the index
            df.reset_index(inplace=True)
            df.drop(columns=["index"], inplace=True)

            #creating new columns for centroid information
            df["x"] = df["left"] + df["width"]
            df["y"] = df["top"] + df["height"]
```

DataFrame Pre Processor

Step 4 : Process the text extracted

- Converting whole text to lower case.
- Finding and replacing each date instance to fit in pythons datetime format.

```
importing Jupyter notebook from textPreProcessor.ipynb
Original Date ----> After Processing

5/12/22       ---->      5-12-2022
2/2/2023       ---->      2-2-2023
5:12-2023       ---->      5-12-2023
5,May,2023       ---->      5-05-2023
5 May,2023       ---->      5-05-2023
October 24,2022       ---->      10-24-2022
importing Jupyter notebook from imagePreProcessor.ipynb
```

Date instance pre processor

Step 4 : Finding neighbour elements for each word on invoice

- For invoice date field, relevent neighbour fields chosen were, ["date", "dated", "invoice", "delivery", "order", "due", "payment", "tax", "bill", "receipt", "issue"]
- All the words existing in close proximity of existing word and all the words to the left in the same line were considered as neighbours.
- Each neighbour was a given a column in dataframe, and its existence/non-existence was shown by 1/0.

```
In [8]: def allocateNeighbours(df, neighbourFields):
            for index in df.index:
                x1 = df["x"][index]
                y1 = df["y"][index]
                block = df["block_num"][index]
                para = df["par_num"][index]
                neighbours = []
                for index2 in df.index:
                    x2 = df["x"][index2]
                    y2 = df["y"][index2]
                    block2 = df["block_num"][index2]
                    para2 = df["par_num"][index2]
                    dist = returnDist(x1, y1, x2, y2)
                    if dist<100:
                        neighbours.append(df["text"][index2].lower())
                    if abs(y1-y2)<50 and (x2-x1)<10 and abs(block2-block)<=1 and abs(para2==para)<=1:
                        neighbours.append(df["text"][index2].lower())
                for n in neighbourFields:
                    if n in neighbours:
                        df[n][index]=1
```

Neighbour information allocation

Step 5 : Extracting only rows having a date as text from whole DataFrame

```
In [10]: def findDateDF(dates, dateDF):
             indexes = []
             for date in dates:
                 i = dateDF[dateDF["text"]==date].index
                 for index in i:
                     indexes.append(index)
             return indexes
```

```
In [11]: def dropIndexes(indexes, df):
             indexesToDrop=[]
             for index in df.index:
                 if index not in indexes:
                     indexesToDrop.append(index)
             df.drop(indexesToDrop, inplace=True)
```

Dataset Extraction

Step 6 : These steps were performed on a folder containing several documents to prepare the final dataset for Invoice Date Extraction model

- Resulting DataFrame looked like this,

```
In [21]: resultDF.head()
```

Out[21]:

	block_num	par_num	line_num	word_num	left	top	width	height	conf	text	...	delivery	order	due	payment	tax	bill	receipt	issue
0	8	1	1	6	1343	520	150	23	67.342278	14-Mar-23	...	0	0	0	0	0	0	0	0
1	15	1	2	3	1411	171	100	19	56.832672	4/11/2023	...	0	0	0	0	1	0	0	0
2	9	3	1	3	588	1037	84	19	94.234421	9/10/2022	...	0	0	0	0	0	0	0	0
3	9	3	1	4	745	1031	88	32	84.635925	16-10-2022	...	0	0	0	0	0	0	0	0
4	6	1	1	2	451	289	112	23	92.698341	2-Mar-20	...	0	0	0	0	0	0	0	0

5 rows × 25 columns

Result DataFrame

Step 7 : Now, each entry was manualy annotated 1/0 for is_invoice_date/not_invoice_date

- Images were fairly distributed among both classifications for a better training.

```
In [284]: df["output"].describe()

Out[284]: count    36.000000
          mean      0.555556
          std       0.503953
          min       0.000000
          25%       0.000000
          50%       1.000000
          75%       1.000000
          max       1.000000
          Name: output, dtype: float64
```

Output column visualization

Model Preperation for training :

Step 1 : Dataset preprocessing

- Feature Selection was done and irrelevent features were dropped from the dataset.
- Data was normalized for better results.

```
In [6]: def preProcess(df):
            colsToDrop = ["Unnamed: 0", "left", "top", "width", "height", "imageName", "text"]
            df.drop(columns=colsToDrop, inplace=True)
            df["conf"]=df["conf"]/100
            df["x"]=df["x"]/1000
            df["y"]=df["y"]/1000
```

Dataset Pre Processing

Step 2 : Extracting training and testing sets

- Dataset was split between training and testing sets with a ratio of 80/20.
- 18 features were converted to input, xTrain/Test
  - layout information
  - confidence of prediction
  - neighbour information
- isDate/notDate column was converted to output, yTrain/Test

```
In [115]: def extractData(df):
              train, test = train_test_split(df, test_size=0.2, random_state=42, shuffle=True)
              xTrain = train.drop(columns=["output"]).to_numpy()
              xTest = test.drop(columns=["output"]).to_numpy()
              yTrain = train["output"].to_numpy()
              yTest = test["output"].to_numpy()
              return xTrain, yTrain, xTest, yTest

In [116]: xTrain, yTrain, xTest, yTest = extractData(df)

In [117]: xTrain.shape
Out[117]: (28, 18)

In [118]: xTest.shape
Out[118]: (8, 18)

In [119]: yTrain.shape
Out[119]: (28,)

In [120]: type(xTrain)
Out[120]: numpy.ndarray

In [121]: type(yTrain)
Out[121]: numpy.ndarray
```

Train/Test extraction

Step 3 : Preparing model

- TensorFlow and keras were used to prepare the model.
- Model used was simple ANN, as a complex model would lead to overfitting since the dataset is very small.
    - Optimizer = “adam”
    - Loss = “binary_crossentropy”

```
In [275]: model = keras.Sequential([
              keras.layers.Dense(64, input_shape=(18, ), activation='relu'),
              keras.layers.Dense(32, activation='relu'),
              keras.layers.Dense(1, activation = 'sigmoid')
          ])

In [276]: model.summary()

          Model: "sequential_33"
          _________________________________________________________________
           Layer (type)                Output Shape              Param #
          =================================================================
           dense_148 (Dense)           (None, 64)                1216

           dense_149 (Dense)           (None, 32)                2080

           dense_150 (Dense)           (None, 1)                 33

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

In [277]: model.compile(optimizer='adam',
                        loss="binary_crossentropy",
                        metrics=["accuracy", 'Precision', "Recall", "TruePositives", "TrueNegatives", "FalsePositives", "Fals
```

Model Preperation

- Model was trained for a batch size of 1, and a 100 epochs.

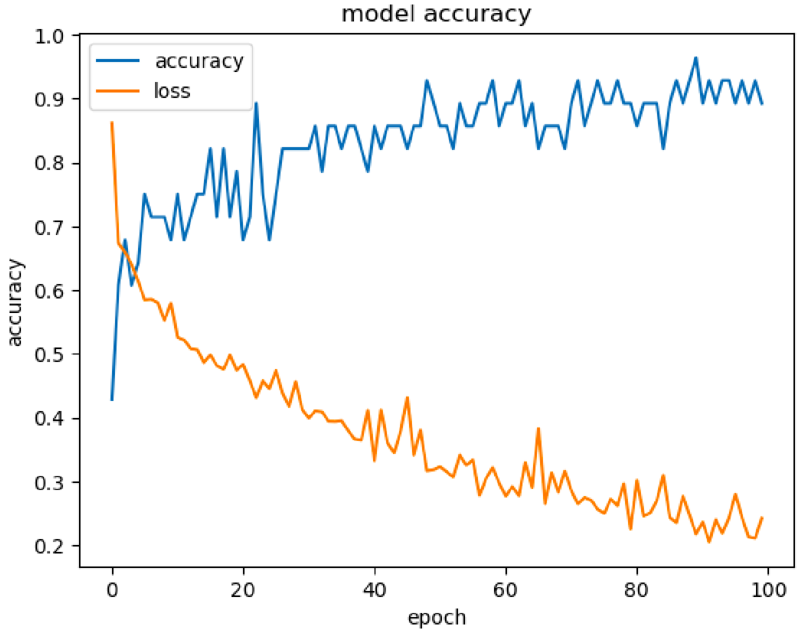
```
In [278]: history = model.fit(xTrain, yTrain,
                    epochs=100, batch_size=1)

          28/28 [==============================] - 0s 1ms/step - loss: 0.2795 - accuracy: 0.8929 - precision: 0.9333 - rec
          all: 0.8750 - true_positives: 14.0000 - true_negatives: 11.0000 - false_positives: 1.0000 - false_negatives: 2.0
          000
          Epoch 97/100
          28/28 [==============================] - 0s 1ms/step - loss: 0.2429 - accuracy: 0.9286 - precision: 0.9375 - rec
          all: 0.9375 - true_positives: 15.0000 - true_negatives: 11.0000 - false_positives: 1.0000 - false_negatives: 1.0
          000
          Epoch 98/100
          28/28 [==============================] - 0s 1ms/step - loss: 0.2129 - accuracy: 0.8929 - precision: 0.9333 - rec
          all: 0.8750 - true_positives: 14.0000 - true_negatives: 11.0000 - false_positives: 1.0000 - false_negatives: 2.0
          000
          Epoch 99/100
          28/28 [==============================] - 0s 979us/step - loss: 0.2105 - accuracy: 0.9286 - precision: 0.9375 - r
          ecall: 0.9375 - true_positives: 15.0000 - true_negatives: 11.0000 - false_positives: 1.0000 - false_negatives:
          1.0000
          Epoch 100/100
          28/28 [==============================] - 0s 1ms/step - loss: 0.2418 - accuracy: 0.8929 - precision: 0.9333 - rec
          all: 0.8750 - true_positives: 14.0000 - true_negatives: 11.0000 - false_positives: 1.0000 - false_negatives: 2.0
          000
```

Model Training

Step 4 : Visualizing the training and evaluating the model.

- Accuracy and loss were plotted.



Training Plot

- Evaluation was done on testing set.

```
In [280]: model.evaluate(xTest, yTest)

          1/1 [==============================] - 0s 243ms/step - loss: 0.4366 - accuracy: 0.8750 - precision: 0.8000 - recal
          l: 1.0000 - true_positives: 4.0000 - true_negatives: 3.0000 - false_positives: 1.0000 - false_negatives: 0.0000e+0
          0

Out[280]: [0.4366227388381958, 0.875, 0.800000011920929, 1.0, 4.0, 3.0, 1.0, 0.0]
```

Testing Results

  - Accuracy = 87.5%
  - Precision = 0.8
  - Recall = 1.0
  - True values = 7/8
  - False values = 1/8

Model Inference on unseen documents :

Step 1 : Preprocessing the document to fit our model.

- Converting to image
- Preprocess image
- Apply OCR using tesseract
- Preprocessing text extracted for relevent extraction field(Invoice Date)
- Normalizing and preprocessing the data extracted

Step 2 : Passing this processed data through our model for output.

- Loading the model
- Processing data for input
- Analyzing output to predict final result
-

```
In [2]: def preProcessImage(image):
            gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
            blur = cv2.GaussianBlur(gray, (3,3),1)
            threshold_img = cv2.adaptiveThreshold(blur,255,1,1,11,2)
            return threshold_img
```

```
In [8]: def processForInput(df):
            colsToDrop = ["left", "top", "width", "height", "text", "index"]
            d = df.drop(columns=colsToDrop)
            d["conf"]=d["conf"]/100
            d["x"]=d["x"]/1000
            d["y"]=d["y"]/1000
            return d.to_numpy()
```

```
In [56]: def predict(filePath):
             f = open(filePath, 'rb')
             readpdf = PyPDF2.PdfReader(f)
             totalpages = len(readpdf.pages)
             if totalpages==1:
                 image = convert_from_path(filePath)
             else:
                 return
             image = np.array(image[0])
             image = preProcessImage(image)
             print(type(image))
             data = pytesseract.image_to_data(image, output_type=Output.DATAFRAME)
             df = pd.DataFrame(data)
             preProcessDataFrame(df)
             dateDF, dates = extractDateDataFrame(df)
             addNeighbours(df)
             indexes = findDateDF(dates, dateDF)
             resultDF = df.copy()
             dropIndexes(indexes, resultDF)
             resultDF.reset_index(inplace=True)
             texts = []
             inputs = []
             for index in resultDF.index:
                 texts.append(resultDF["text"][index])
                 inputs.append(processForInput(resultDF))
             predictions = {}
             model = load_model("/home/aman/Documents/Tally/DocumentAI/Code/Tesseract-Model/Models/tesseractv1.h5")
             for i in range(len(inputs)):
                 text = texts[i]
                 input = inputs[i]
                 pred = model.predict(input)
                 predictions[text]=max(pred)
             if len(predictions)==0:
                 return None
             res = max(zip(predictions.values(), predictions.keys()))[1]
             return res
```

Inference Results

- Model was able to predict correct output on many of the unseen images.
- It was not able to predict correct date as an invoice date on some images.
- It was not able to extract any date fields on some images.

```
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 25ms/step
31-12-22
```

Correct Output

```
<class 'numpy.ndarray'>
<string>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy
<string>:20: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy
None
```

Unable to detect any date fields

Important Findings and observations

- Tesseract is very fast as compared to EasyOCR.
- Tesseract is a c++ library and directly calls the c++ code in pyhon script.

```
pytesseract.pytesseract.tesseract_cmd="/home/aman/anaconda3/envs/tallyInvoiceParser.env/bin/tesseract"
os.environ['TESSDATA_PREFIX'] = "/home/aman/anaconda3/envs/tallyInvoiceParser.env/share/tessdata"
os.environ['MLIR_CRASH_REPRODUCER_DIRECTORY']='tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269'
```

- Output of tesseract includes a lot of information which helps in preparing NER models like this.
- Output was incorrect in some cases due to bad approach used for neighbour information extraction.(Use of dynamic radius rather than static would resolve this issue)
- Date was not correctly extracted in some cases due to poor preprocessing of text for date.(Combination of several models like spacy lg, spacy sm and python's datetime can improve this issue)

Further Improvements

- Accuracy can be further improved by ,
    - Increasing the size of dataset.
    - Using different normalization techniques.
    - Using a better approach to allocate neighbours, rather than setting a fixed radius.
    - Rather than setting 'radius to find neighbours' as a 'constant', it must be set as a 'ratio of page size', since page sizes are not same for each document.
        - Example :
            - radius = 10   #static for all images
            - radius = image.shape[0]//100   #dynamic
- Similar models can be prepared for other fields to be extracted, by using same model architecture but different preprocessing techniques.
- All these trained models can be combined to perform our task of converting Unstructured Data to Structured Data.