Indian Institute of Technology Bombay

Report for

PH303 : Supervised Learning Project

Semantic Table, Column & Cell Recognition through Image Segmentation using TableNet

Submitted by:

Kandarp Solanki

Supervisor:

Prof. Biplab Banerjee

Abstract

This supervised learning project harnesses the power of **DenseNet** and **TableNet** for precise table, column, and cell masking through image segmentation. The project's focus lies in automating the extraction of structured data from diverse unstructured documents. By utilizing the advanced capabilities of DenseNet and Tablenet in table detection and segmentation, we aim to enhance the precision of table extraction. This research has diverse applications, including data mining, content analysis, and information retrieval, contributing to more efficient decision-making and knowledge extraction in our data-centric world.

Contents

Chapter 1

Introduction

TableNet is a deep learning model that uses a common network to solve two major issues:

- Detection of tabular areas in a given image **(Table detection)**
- Extraction of information from the rows and columns of the detected table **(Table Structure Recognition)**

The original **TableNet** model emphasises only on table and column extraction but this project involved further extending this idea to cell extraction. Given the interdependence of these tasks doing it with a single neural network is considered beneficial.

The architecture of the model is very elegant, similar to an *encoder-decoder* model with the *encoder* usually encoding the positions and structural aspects of the table and the *decoder* uses this information for accurate mask generation

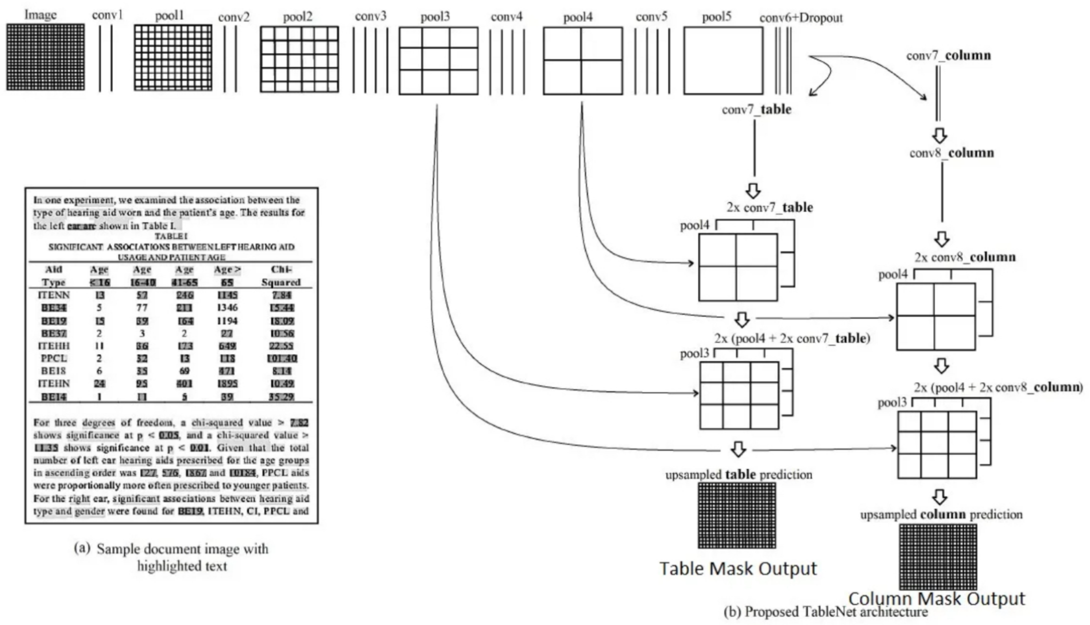


Image
conv1
pool1
conv2
pool2
conv3
pool3
conv4
pool4
conv5
pool5
conv6+Dropout
conv7_column
conv7_table
conv8_column
2x conv7_table
pool4
2x conv8_column
pool4
2x (pool4 + 2x conv7_table)
pool3
2x (pool4 + 2x conv8_column)
pool3
upsampled table prediction
Table Mask Output
upsampled column prediction
Column Mask Output
(a) Sample document image with highlighted text
(b) Proposed TableNet architecture

In one experiment, we examined the association between the type of hearing aid worn and the patient's age. The results for the left ear are shown in Table I.

TABLE I
SIGNIFICANT ASSOCIATIONS BETWEEN LEFT HEARING AID USAGE AND PATIENT AGE

Aid Type	Age ≤16	Age 16-40	Age 41-65	Age > 65	Chi-Squared
ITENN	13	57	246	1145	7.84
BE34	5	77	211	1346	15.44
BE19	15	39	164	1194	18.09
BE37	2	3	2	27	10.56
ITEHH	11	36	173	649	22.55
PPCL	2	32	13	118	101.40
BE18	6	35	69	471	8.14
ITEHN	24	95	401	1895	10.49
BE14	1	11	5	39	35.29

For three degrees of freedom, a chi-squared value > 7.82 shows significance at $p < 0.05$, and a chi-squared value > 11.35 shows significance at $p < 0.01$. Given that the total number of left ear hearing aids prescribed for the age groups in ascending order was 127, 576, 1867 and 10184, PPCL aids were proportionally more often prescribed to younger patients. For the right ear, significant associations between hearing aid type and gender were found for BE19, ITEHN, CI, PPCL and

Fig. DenseNet Model Architecture

Chapter 2

Dataset and its preprocessing

2.1 Hardwares and Softwares Used

Due to hardware limitations, we worked with **Google Colab**, which is an online collaborative platform, for the entire project which gives access to free usage of **T4 GPU** on Google's computing platforms for faster computations. The data was accessed from my personal Google Drive everytime the code is run so that the code is accessible to all.

2.2 Marmot Dataset

The dataset used in this entire project is the **Marmot Table Recognition Data** which is used as a table detection dataset but does not contain ground truth[1] values for column and cell detection. The original Marmot dataset contains 1016 images in total out of which 509 are English documents and rest are in Chinese (we use only the **English** one).

NOTE: This dataset was tried for table and column detection only whereas the cell detection was carried out using data provided to us by **PerpetualBlock**[2]

2.3 Format of the Data

The data for table and column detection included images of **.bmp** (Bitmap Image File) and data in XML file for table and column coordinates

The data sent by the company was a text file in the following format for each image:

{0,1,2,3} {x-coordinate} {y-coordinate} {x-width} {y-height}

All these parameters were given to us in **relative terms** i.e how much fraction (e.g 0.125) w.r.t the height and width of the given image. {0,1,2,3} corresponds to Cell, Column, Description, Table respectively.

[1]information that is known to be real or true, as opposed to information by inference

[2]It is a **Partex** company who funded us for this project

2.4 Pre-Processing

For Table/Column Masking

The data available to us was in a crude format so it required some pre-processing before it could be converted into a user-friendly format.

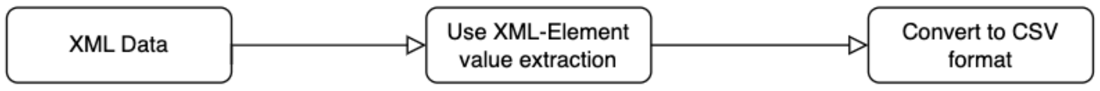


Fig. How data is processed?

An **XML** (eXtensible Markup Language) file is a text-based document that uses tags to structure data for storage and transport.

The XML file given to us contains a lot of information about the data/image and we do not need all of it. We only need a certain branch or root of the entire data structure which contains information about the bounding boxes i.e the four coordinates within which the tables and columns are present. While doing this we also need to make sure that the image is reshaped to a fixed scale which is 1024x1024 in our case. All the bounding boxes are thus made w.r.t to these new coordinates after scaling

As an example, imagine if the size of the image below doubles in both dimensions, the mask sizes happen to have the same effect

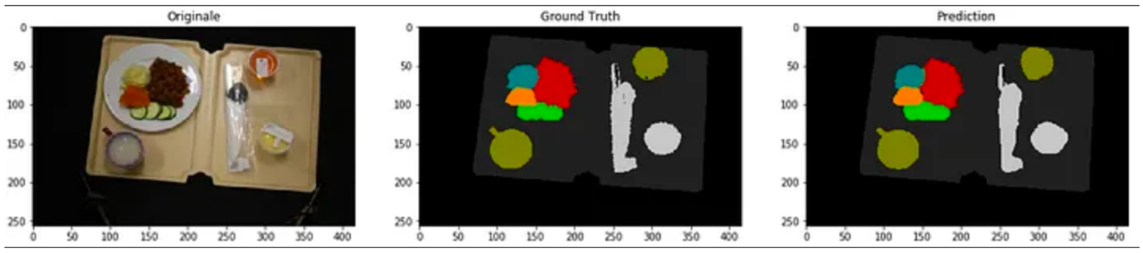


Figure 2.1: A typical example of masking, its ground truth and prediction

For Cell Masking

Since the data given to us for the cell masking was in a different format as mentioned earlier, we had to write our own python script to generate a *csv* file which contained the organised information which is needed by us. The code for the script is attached in the **Appendix**.
Take a note that all the file address while saving the *csv* file were given the same as that of the drive as these paths will be used to access it when using **Google Colab**.

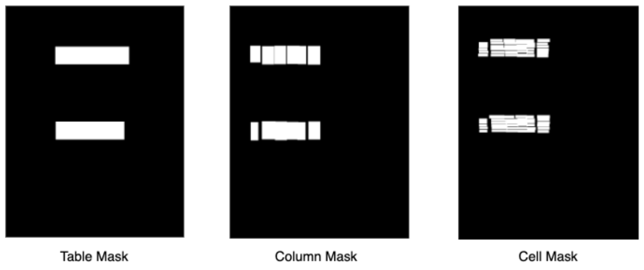


Figure 2.2: Table, Column and Cell Masks for a given table generated using the script

Chapter 3

Model Pipeline

3.1 Definitions

Here is a list of some common terms/methodologies that are used in the model pipeline:

- **Convolutions** - Convolution refers to reducing the dimension of the overall input such that each new pixel now resembles a new collective value of the old pixels. The reverse process is known as **deconvolution**, although it is lossy in nature.
- **Kernels** - Kernels can be considered like a moving matrix used while convolutions. It assigns the center pixel the weighted dot product of the kernel matrix with the sub-region of input data
- **Strides** - Strides define how many steps the kernel moves during a convolution
- **Dropout** - A Dropout layer is a mask that neglects/discards a fraction of values from some of the neurons
- **Pooling** - Pooling of parameters refers to reducing the size of feature maps to reduce the complexity of computation and make it faster
- **Activation Functions** - Decides whether the neuron should be activated or not and upto what extent using a function of weighted sum and adding a bias term. It introduces **non-linearity**
- **Batch Normalization** - Addition of extra layers in the neural network to make it faster by performing some standardizing and normalising operations

3.2 Model Architecture

Encoder

For the encoder part, we used the pretrained **DenseNet121** model. The weights of the DenseNet121 model are set as non-trainable and will not be changed using training.

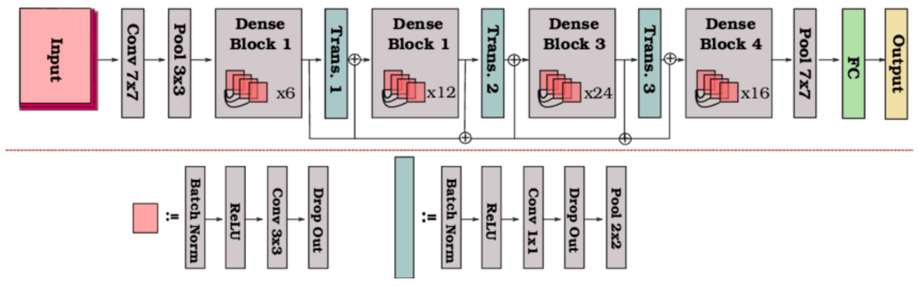


Figure 3.1: Encoder Pipeline of pre-trained DenseNet model

The summary of the encoder and decoder parts of the model is as below:

```
|------------------------------|-------------------------|-----------|
| Layer (type)                 | Output Shape            | Param #   |
|------------------------------|-------------------------|-----------|
| DenseNet                     |                         |           |
| features                     | (256, 32, 32)           | 0         |
| densenet_out_1 (Sequential)  | (256, 64, 64)           | 16,896    |
| densenet_out_2 (Sequential)  | (512, 32, 32)           | 984,576   |
| densenet_out_3 (Sequential)  | (1,024, 32, 32)         | 590,848   |
| TableDecoder                 |                         |           |
| conv_7_table (Conv2d)        | (256, 32, 32)           | 590,080   |
| ReLU (ReLU)                  | (256, 32, 32)           | 0         |
| upsample_1_table (ConvTran   | (128, 64, 64)           | 131,200   |
| ReLU (ReLU)                  | (128, 64, 64)           | 0         |
| upsample_2_table (ConvTran   | (256, 128, 128)         | 491,776   |
| ReLU (ReLU)                  | (256, 128, 128)         | 0         |
| upsample_3_table (ConvTran   | (1, 1024, 1024)         | 4,097     |
| ColumnDecoder                |                         |           |
| conv_8_column (Sequential)   | (256, 32, 32)           | 753,408   |
| ReLU (ReLU)                  | (256, 32, 32)           | 0         |
| upsample_1_column (ConvTra   | (128, 64, 64)           | 131,200   |
| ReLU (ReLU)                  | (128, 64, 64)           | 0         |
| upsample_2_column (ConvTra   | (256, 128, 128)         | 491,776   |
| ReLU (ReLU)                  | (256, 128, 128)         | 0         |
| upsample_3_column (ConvTra   | (1, 1024, 1024)         | 4,097     |
| CellDecoder                  |                         |           |
| conv_9_cell (Sequential)     | (256, 32, 32)           | 753,408   |
| ReLU (ReLU)                  | (256, 32, 32)           | 0         |
| upsample_1_cell (ConvTrans   | (128, 64, 64)           | 131,200   |
| ReLU (ReLU)                  | (128, 64, 64)           | 0         |
| upsample_2_cell (ConvTrans   | (256, 128, 128)         | 491,776   |
| ReLU (ReLU)                  | (256, 128, 128)         | 0         |
| upsample_3_cell (ConvTrans   | (1, 1024, 1024)         | 4,097     |
| TableNet                     |                         |           |
| base_model (DenseNet)        | (1, 1024, 32, 32)        | 0         |
| conv6 (Sequential)           | (256, 32, 32)           | 328,704   |
| ReLU (ReLU)                  | (256, 32, 32)           | 0         |
| table_decoder (TableDecode   | (1, 1024, 1024)         | 1,242,081 |
| column_decoder (ColumnDeco   | (1, 1024, 1024)         | 1,242,081 |
| cell_decoder (CellDecoder)   | (1, 1024, 1024)         | 1,242,081 |
| Total Parameters             |                         | 7,774,355 |
|------------------------------|-------------------------|-----------|
```

Figure 3.2: Model Summary

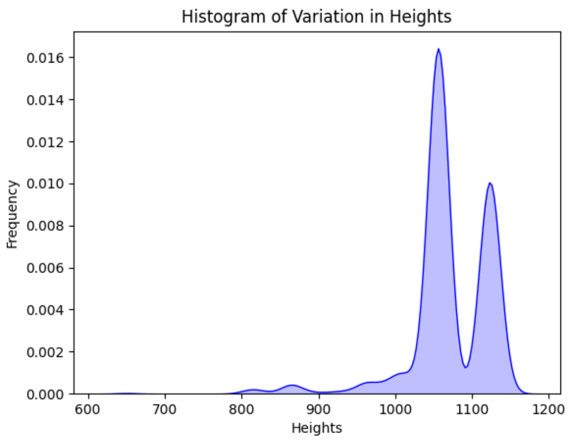
Chapter 4

Observations

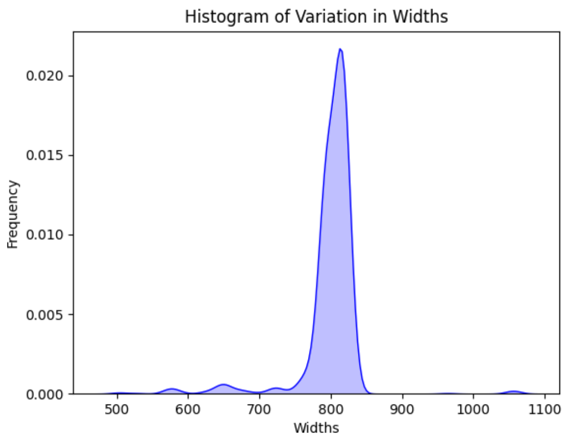
4.1 Exploratory Data Analysis (EDA)

Table and Column Encoder (first two) and **Cell Encoder** (later two)

Below are the plots for kernel density estimates of the histogram for the variation in heights and widths across the dataset:

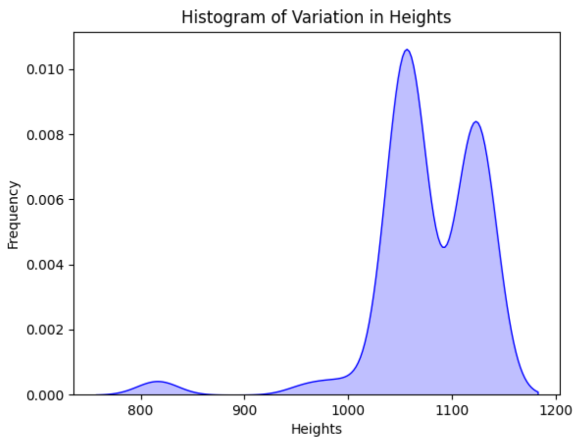


(a) Frequency v/s Height

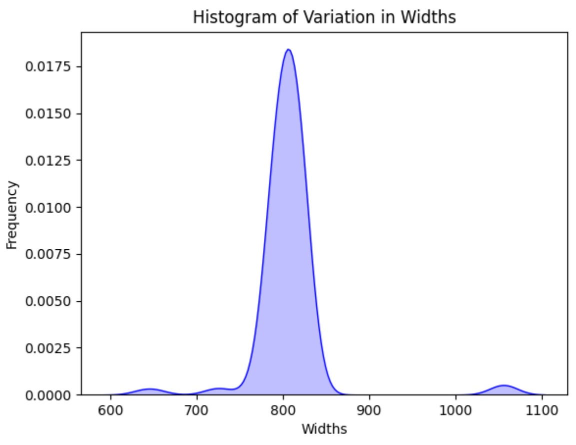


(b) Frequency v/s Width

Figure 4.1: KDE plots from Marmot Dataset



(a) Frequency v/s Height



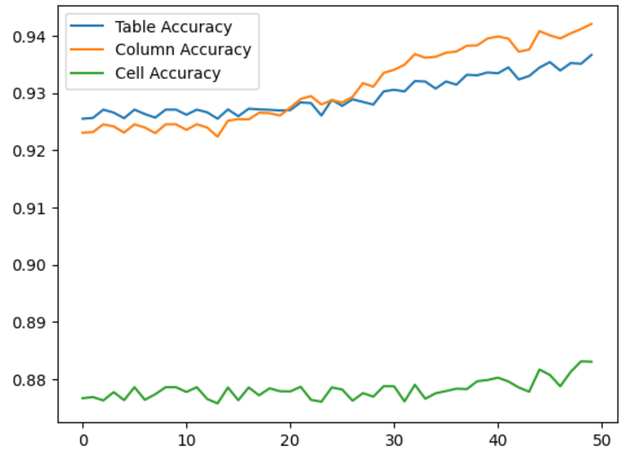
(b) Frequency v/s Width

Figure 4.2: KDE plots from Dataset by PerpetualBlock

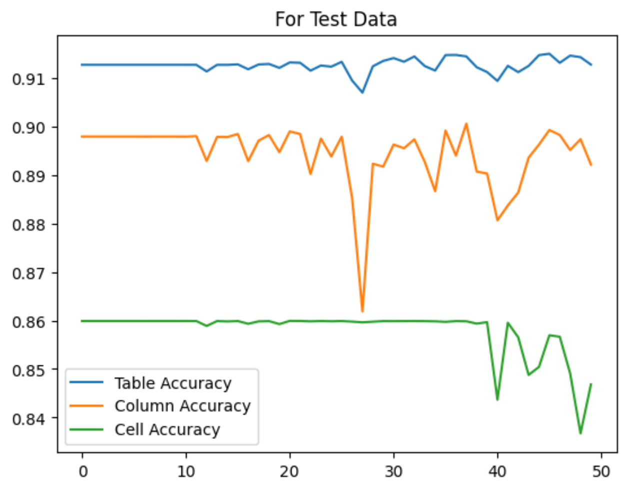
4.2 Training & Test Metrics

Here are a few plots which depict the accuracy, loss over training and test data:

4.2.1 Accuracy

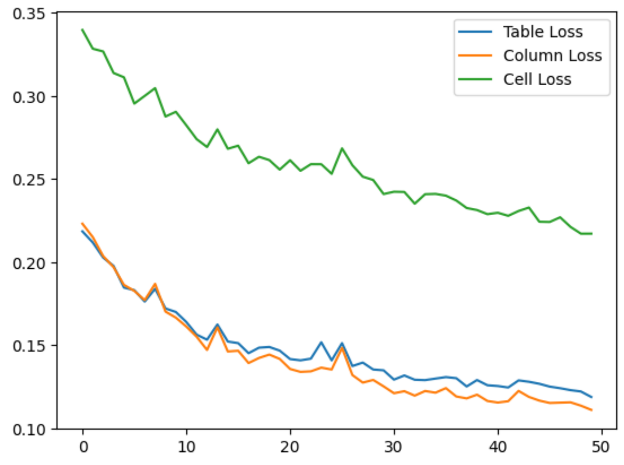


(a) Training Accuracy

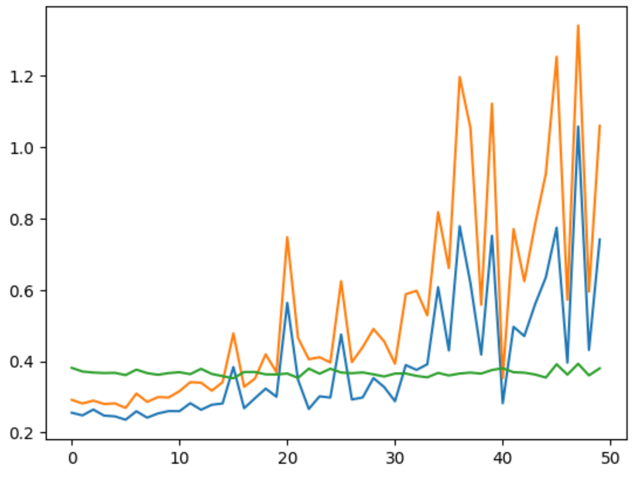


(b) Test Accuracy

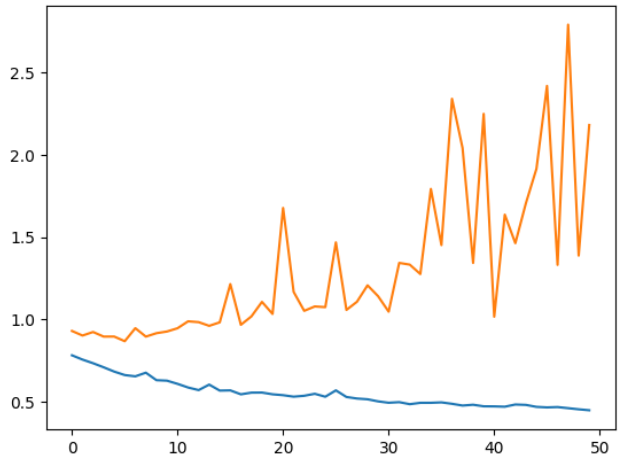
4.2.2 Loss



(a) Training Loss



(b) Test Loss

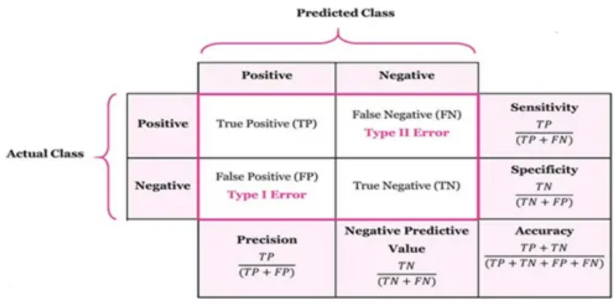


(a) Combined Loss

Chapter 5

Statistical Analysis & Future Goals

5.1 The Confusion Matrix



		Predicted Class		
		Positive	Negative	
Actual Class	Positive	True Positive (TP)	False Negative (FN) Type II Error	Sensitivity $\frac{TP}{(TP+FN)}$
	Negative	False Positive (FP) Type I Error	True Negative (TN)	Specificity $\frac{TN}{(TN+FP)}$
		Precision $\frac{TP}{(TP+FP)}$	Negative Predictive Value $\frac{TN}{(TN+FN)}$	Accuracy $\frac{TP+TN}{(TP+TN+FP+FN)}$

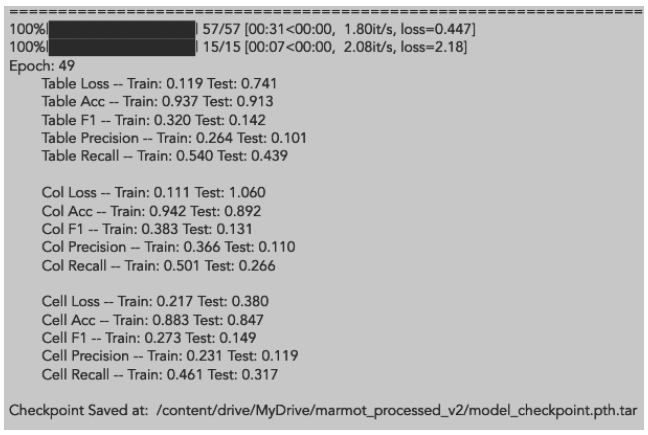
(a) Confusion Matrix

The confusion matrix is a table that is used to define the performance of a classification algorithm.

It has **four** main performance indicators:

- **Accuracy** : Defined as True Positives + True Negatives over all outcomes
- **Sensitivity/Recall** : Defined as True Positives over True Positives + False Negatives
- **Specificity** : Defined as True Negatives over True Negatives + False Positives
- **Precision** : Defined as True Positives over True Positives + False Positives

Here is how our model performed on it:



```
================================================================
100%|████████████| 57/57 [00:31<00:00,  1.80it/s, loss=0.447]
100%|████████████| 15/15 [00:07<00:00,  2.08it/s, loss=2.18]
Epoch: 49
      Table Loss -- Train: 0.119 Test: 0.741
      Table Acc -- Train: 0.937 Test: 0.913
      Table F1 -- Train: 0.320 Test: 0.142
      Table Precision -- Train: 0.264 Test: 0.101
      Table Recall -- Train: 0.540 Test: 0.439

      Col Loss -- Train: 0.111 Test: 1.060
      Col Acc -- Train: 0.942 Test: 0.892
      Col F1 -- Train: 0.383 Test: 0.131
      Col Precision -- Train: 0.366 Test: 0.110
      Col Recall -- Train: 0.501 Test: 0.266

      Cell Loss -- Train: 0.217 Test: 0.380
      Cell Acc -- Train: 0.883 Test: 0.847
      Cell F1 -- Train: 0.273 Test: 0.149
      Cell Precision -- Train: 0.231 Test: 0.119
      Cell Recall -- Train: 0.461 Test: 0.317

Checkpoint Saved at:  /content/drive/MyDrive/marmot_processed_v2/model_checkpoint.pth.tar
```

(a) Performance of our Model

Although it did not perform fairly well on a few indicators, it did so very fairly when it comes to accuracy. We might some fine tuning of our hyperparameters in order to achieve a better stability to the results.

Moreover, the amount of data over which we worked on was significantly lesser (about $\tilde{1}0$ times less) than what the **Marmot** dataset offers and still the model could achieve upto 93%, 94% and 88% accuracy for Table, Column and Cell detection respectively.

5.2 Future Plans

In future, we aim to develop a more robust model pipeline that simultaneously enhances performance across all three verticals. Achieving balanced accuracy for these tasks can be challenging with the same dataset, but we believe that through improved neural network designs and proper data segmentation, we can achieve promising results even with a smaller dataset. Our goal is to create specialized neural network architectures tailored to the unique requirements of each vertical, allowing us to extract maximum value from the available data and deliver superior results in the future.

Acknowledgements

I would like to express my heartfelt gratitude to **Prof. Biplab Banerjee** for his invaluable guidance and unwavering support throughout the course of this project in the domain of Semantic Table, Column & Cell Recognition through Image Segmentation using TableNet. His profound expertise and mentorship have been instrumental in shaping this research endeavor and have contributed significantly to its success. I am truly fortunate to have had the privilege of working under his mentorship, which has enriched my understanding and passion for this field. Also, a huge thanks to **Mr. Mohd Hasan** from CMiNDS who helped us with some debugging issues and guided us during the project.

Appendix

Python Script for Data Processing (Cell Masking)

```
from PIL import Image
import csv
import pandas as pd

files = ["....all file names come here...."]

n = 1
temp_list = []
for i in files:
    mask_path = f"/Users/kandarpsolanki/Downloads/SLP/labels/{i}.txt"

    img_path = f"/Users/kandarpsolanki/Downloads/SLP/images/{i}.bmp"

    image = Image.open(img_path)
    (width,height) = image.size #(width,height)

    img_path2 = f"/content/drive/MyDrive/marmot_processed_v2/image/{i}.bmp"


    mask_coords = open(mask_path)

    table_bboxes = []
    col_bboxes = []
    cell_bboxes = []

    for x in mask_coords:
        info = x.split()
        if int(info[0]) == 0:
            x_left_up = int(width*float(info[1]))
            y_left_up = int(height*float(info[2]))
            x_right_down = int(x_left_up + width*float(info[3]))
            y_right_down = int(y_left_up + height*float(info[4]))

            cell_bboxes.append([x_left_up,y_left_up,x_right_down,y_right_down])

        if int(info[0]) == 1:
            x_left_up = int(width*float(info[1]))
            y_left_up = int(height*float(info[2]))
            x_right_down = int(x_left_up + width*float(info[3]))
            y_right_down = int(y_left_up + height*float(info[4]))

            col_bboxes.append([x_left_up,y_left_up,x_right_down,y_right_down])


        if int(info[0]) == 3:
            x_left_up = int(width*float(info[1]))
            y_left_up = int(height*float(info[2]))
            x_right_down = int(x_left_up + width*float(info[3]))
            y_right_down = int(y_left_up + height*float(info[4]))

            table_bboxes.append([x_left_up,y_left_up,x_right_down,y_right_down])


    table_mask_path = f"/content/drive/MyDrive/marmot_processed_v2/table_mask/{i}.png"
    col_mask_path = f"/content/drive/MyDrive/marmot_processed_v2/col_mask/{i}.png"
    cell_mask_path = f"/content/drive/MyDrive/marmot_processed_v2/cell_mask/{i}.png"
    if len(table_bboxes) > 0:
        hastable = 1
    else:
        hastable = 0

    temp_list.append([img_path2, table_mask_path, col_mask_path, cell_mask_path,
    height, width, hastable, len(table_bboxes), len(col_bboxes), len(cell_bboxes),
    table_bboxes, col_bboxes, cell_bboxes])


df = pd.DataFrame(temp_list, columns=['img_path','table_mask','col_mask','cell_mask',
'original_height','original_width','hasTable','table_count','col_count','cell_count',
'table_bboxes', 'col_bboxes','cell_bboxes'])

df.to_csv("processed_data_1.csv", index=False)
print('Done!')
```

Python Script for Histogram Generation

```
import csv
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

file = open('processed_data_1.csv')

data = csv.reader(file)

heights = []
widths = []
tables = []
columns = []
cells = []
```

```
header = next(data)
for row in data:
    try:
        heights.append(int(row[3]))
    except:
        heights.append(0)
    try:
        widths.append(int(row[4]))
    except:
        widths.append(0)
    try:
        tables.append(int(row[6]))
    except:
        tables.append(0)
    try:
        columns.append(int(row[7]))
    except:
        columns.append(0)
    try:
        cells.append(int(row[9]))
    except:
        cells.append(0)


tables = np.array(tables)
columns = np.array(columns)
cells = np.array(cells)
labels, counts = np.unique(cells, return_counts=True)
plt.bar(labels, counts, align='center')
# plt.gca().set_xticks(labels)
plt.xticks(rotation=90)

# plt.hist(columns, bins = 25)
# sns.kdeplot(heights, shade=True, color='blue')

# Add labels and a title
plt.xlabel('No. of Cells')
plt.ylabel('Frequency')
plt.title('Histogram of Number of Cells')

# Display the histogram
plt.show()
```

Bibliography

Ayush M., 2022, *Table-Extraction*, https://github.com/ayushm380/Table-Extraction

ES L., 2022, *TableNet using PyTorch*, https://github.com/LidorPrototype/TableNetTable2df

Fetahu B., Anand A., Koutraki M., 2019, *TableNet: An Approach for Determining Fine-grained Relations for Wikipedia Tables* (arXiv:1902.01740)

GIMP, *Software for data visualisation*

Paliwal S., D V., Rahul R., Sharma M., Vig L., 2020, *TableNet: Deep Learning model for end-to-end Table detection and Tabular data extraction from Scanned Document Images* (arXiv:2001.01469)