**Steps and Links of TableNet Model**

With the increase use of mobile devices, customers tend to share documents as images rather than scanning them. These images are later processed manually to get important information stored in Tables. These tables can be of different sizes and structures. It is therefore expensive to get information in Tables from images.

With TableNet we will employ and end-to-end Deep learning architecture which will not only localize the Table in an image, but will also generate structure of Table by segmenting columns in that Table.

We will use both Marmot and Marmot Extended dataset for Table Recognition. Marmot dataset contains Table bounding box coordinates and extended version of this dataset contains Column bounding box coordinates.

Marmot Dataset : <https://www.icst.pku.edu.cn/cpdp/docs/20190424190300041510.zip>

Marmot Extended dataset : <https://drive.google.com/drive/folders/1QZiv5RKe3xlOBdTzuTVuYRxixemVIODp>

Download processed Marmot dataset: <https://drive.google.com/file/d/1irIm19B58-o92IbD9b5qd6k3F31pqp1o/view?usp=sharing>

**Model GitHub repo is in:**

[GitHub - asagar60/TableNet-pytorch: Pytorch Implementation of TableNet](https://github.com/asagar60/TableNet-pytorch)

This github link contains all the links for Downloading the Marmot Datasets which we are using to run our code. This link also contain links for medium vlogs for this research article. You will also get link for saved pretrained model (DenseNet121) which we have use to train our model.

* Training folder you can find all the scripts which will run on background.
* You have to run EDA-v1 python file in the link which you have to run for EDA analysis and for generating [processed\_data.csv](https://github.com/asagar60/TableNet-pytorch/blob/main/processed_data.csv) which contain Image path, table mask path, column mask path , table and column boundary boxes.
* After that you have to run Output python file for training and testing your model.

We have around 994 images documents from which we will be dealt using **semantic segmentation** by predicting pixel-wise regions of Table and columns in them.

Image data is in .**bmp** (bitmap image file) format and bounding box coordinates are in **XML** files following **Pascal VOC** format.

# Steps for Data Pre-Processing

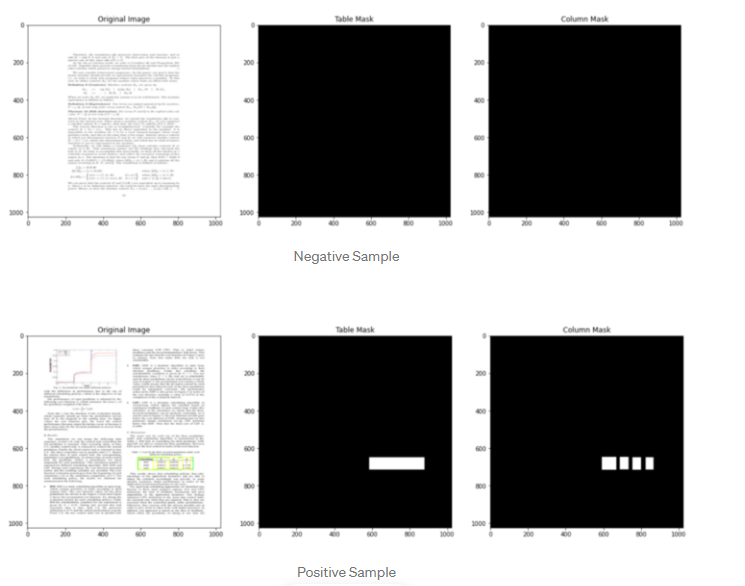
Image data is in .**bmp** (bitmap image file) format and bounding box coordinates are in **XML** files following **Pascal VOC** format.

First we define 3 utility functions

* **get\_table\_bbox()**: This function will extract Table Coordinates using xml file from original marmot dataset and scale them w.r.t to new image shape
* **get\_col\_bbox()**: This function will extract Column Coordinates using xml file from extended marmot dataset and scale them w.r.t to new image shape, and if no table coordinates are returned from **get\_table\_bbox()**function, we will approximate them using column bounding boxes.
* **create\_mask()** : This function takes in bounding boxes ( table / column) and creates mask with 1 channel. If no bounding boxes are given, it creates an empty mask.

Basic idea of preprocessing:

* Read image file, table\_xml and column\_xml.
* Resize image to (1024, 1024) and convert them to RGB ( if not already)
* Get both table and column bounding box
* Create mask for both
* Save Image and mask to disk
* converting processed\_data to csv file.
* Let’s check the masks that were created based on table and column coordinates



**Steps to Train and Test the Model**

# We will use DenseNet121 as encoder and build model upon it.

# 

# Path- "/home/ec2-user/SageMaker/Ayush/TableNet-pytorch-main “

### Trainable Params

### 

We constrained the training **epochs to 50–100**, and tried different models for encoders.

Densenet121 worked best as encoder compared to VGG19, ResNet-18 and EfficientNet. It is worth mentioning that performance of ResNet-18 and EfficientNet was almost close to DenseNet, but I chose the model based on Best F1 Score on Test data.

Test function takes data loader, model and loss as input and returns F1 Score, Accuracy, Precision, Recall and Loss for that epoch.

# 

# 

# Model testing result: - Model testing will generate the table and column mask, using this masks we can crop the table image and then we can extract the information using Tesseract OCR, which provided by AWS.

# 

# After this we will crop the mask image using the boundary boxes.

# 

# Once we have Crop image, we can extract the information using Tesseract OCR, which provided by AWS.

# 

## **Observations**

* we have observed that Bad / worst predictions are given by images with **colored tables**. Model didn't predict anything and F1 score is close to 0.0. There are very few images in the dataset which have colored tables.
* Good predictions come from those images which predicted good Table mask, but it also predicted columns in the table where in actual there were no columns.
* Best Predictions are images which helped model learn table and column boundaries even without line demarcations

**Fixing Image Problems**

We have 2 options, which might improve model performance,

* Remove colored images, or *[ Problem: Data reduction is an issue here as we already have less data]*
* We can have uniform data by converting all images to grayscale first and then increase the number of channels in preprocessing, and Train model again.

**Improving model predictions using OpenCV2**

We can still see uneven boundaries of predicted table and column masks. In some cases, Table mask predictions are not even filled inside. If we directly crop the mask portions of the image to get Table, we might lose some information. Not to mention, there are other areas with activations in the predicted table mask (which are not tables).

To solve these issues, we will use **contours**from classical image processing techniques.

*Basic Idea*:

* Get contours around the activation from the predicted table mask.
* Remove contours which cant be rectangle / small patch of activations.
* Get bounding coordinates of the remaining contour.
* Repeat the same process with Column Masks

Reference:- [[2001.01469] TableNet: Deep Learning model for end-to-end Table detection and Tabular data extraction from Scanned Document Images (arxiv.org)](https://arxiv.org/abs/2001.01469)