INVESTIGATIVE TECHNIQUES

**Run Length Smoothing Algorithm (RLSA)**

The Run Length Smoothing Algorithm (RLSA) is a method that can be used for Block segmentation and text discrimination. The method developed for the Document Analysis System consists of two steps. First, a segmentation procedure subdivides the area of a document into regions (blocks), each of which should contain only one type of data (text, graphic, halftone image, etc.). Next, some basic features of these blocks are calculated.

The basic RLSA is applied to a binary sequence in which white pixels are represented by 0’s and black pixels by 1’s. The algorithm transforms a binary sequence x into an output sequence y according to the following rules:

1. 0’s in x are changed to 1’s in y if the number of adjacent 0’s is less than or equal to a predefined limit C.

2. 1’s in x are unchanged in y .

When applied to pattern arrays, the RLSA has the effect of linking together neighboring black areas that are separated by less than C pixels. With an appropriate choice of C, the linked areas will be regions of a common data type.

The RLSA is applied row-by-row as well as column-by-column to a document, yielding two distinct bit-maps. Because spacings of document components tend to differ horizontally and vertically, different values of C are used for row and column processing. Bit-maps are then combined in a logical AND operation. Additional horizontal smoothing using the RLSA produces the final segmentation result.

To provide identification for subsequent processing, a unique label is assigned to each block. Simultaneously with block labeling, the following measurements are taken:

-> Total number of black pixels in a segmented block (BC).

-> Minimum x-y coordinates of a block and its x, y lengths (Xmin, Ymin, delX, delY).

-> Total number of black pixels in original data from the block (DC).

-> Horizontal white-black transitions of original data (TC).

These measurements are stored in a table and are used to calculate the following features:

-> The height of each block segment: H = delY.

-> The eccentricity of the rectangle surrounding the block: E = delX/delY.

-> The ratio of the number of block pixels to the area of the surrounding rectangle: S=BC/(delX \* delY). If S is close to one, the block segment is approximately rectangular.

-> The mean horizontal length of the black runs of the original data from each block: Rm=DC/TC.

**Document image layout analysis**

Many methods have been proposed for document image layout analysis.

They can be classified into three different groups:

(i) region or block based classification methods- Segment a document image page into document zones, and then classify them

into meaningful semantic classes

(ii) pixel based classification methods- Takes each pixel individually into account and use a classifier to generate a labelled image with regions hypotheses.

(iii) connected component classification methods- Use local information to create object hypotheses that are further inspected, combined and refined, and finally classified.

When it comes to image classification, convolutional neural networks (CNNs) ) have been widely adopted including document analysis. It inherent very intense computational burden usually limits the cost-benefit of using them in document storage and retrieval applications where low memory and fast processing are vital.

Block based classification method that consists of three steps:

i) pre-process a document input image and segment it into its blocks of content

ii) use their vertical and horizontal projections to train a CNN model for multi-class classification considering text,

image and table classes

iii) detect new documents layout using a pipeline including the trained CNN model.

Methodology

1) Segmenting blocks of content in the document image

i) Single pages are converted into gray-scale images.

ii) Then processed by the nonlinear, run-length smoothing algorithm to detect regions with high chance of containing information. The algorithm is applied in both horizontal and vertical directions and the resulting binary images are combined using the operator AND.

iii) Next, a 3 × 3 dilation operation is performed two times over the resulting binary image to create blocks of content.

iv) Finally, we iteratively detect the largest connected component in the binary image and denote it as a block of content. The detection process continues until no more connected components are found in the image.

2) Classifying blocks of content in the document image

Use a CNN model to classify them into three different classes: text, tables and images.

Two different CNN architectures: a bi-dimensional approach commonly used in different computer vision problems, used as a baseline; and the herein proposed fast one-dimensional architecture that uses onedimension projections to deliver very similar results with much less data usage and processing time.

**FASTER-RCNN**

An RPN(Region Proposal Network) is a fully convolutional network that simultaneously predicts object bounds and objectness scores at each position. The RPN is trained end-to-end to generate high-quality region proposals, which are used by Fast R-CNN for detection. We merge RPN and Fast R-CNN into a single network by sharing their convolutional features

Selective Search greedily merges superpixels based on engineered low-level features. Convolutional feature maps used by region-based detectors, like Fast RCNN, can also be used for generating region proposals. On top of these convolutional features, we construct an RPN by adding a few additional convolutional layers that simultaneously regress region bounds and objectness scores at each location on a regular grid. The RPN is thus a kind of fully convolutional network (FCN) and can be trained end-to-end specifically for the task for generating detection proposals.

Object proposal methods include those based on grouping super-pixels (e.g. Selective Search, CPMC, MCG) and those based on sliding windows (e.g. objectness in windows, EdgeBoxes).

The R-CNN method trains CNNs end-to-end to classify the proposal regions into object categories or background.

R-CNN mainly plays as a classifier, and it does not predict object bounds (except for refining by bounding box regression). Its accuracy depends on the performance of the region proposal module

**Detectron2**

Detectron2 is Facebooks new vision library that allows us to easily use and create object detection, instance segmentation, keypoint detection and panoptic segmentation models. It has a simple, modular design that makes it easy to rewrite a script for another data-set.

It is a PyTorch based modular computer vision model library. It is the second iteration of Detectron, originally written in Caffe2. The Detectron2 system allows you to plug in custom state of the art computer vision technologies into your workflow. Detectron2 includes all the models that were available in the original Detectron, such as Faster R-CNN, Mask R-CNN, RetinaNet, and DensePose. It also features several new models, including Cascade R-CNN, Panoptic FPN, and TensorMask.

PROPOSED SOLUTION

For training object detection model for custom dataset, We need to start with building a model using a Feature Pyramid Network combined with a Region Proposal Network if we opt for region proposal based methods such as Faster R-CNN or we can also use one-shot detector algorithms like SSD and YOLO.Either of them is a bit complicated to work with if we want to implement it from scratch. We need a framework where we can use state-of-the-art models such as Fast, Faster, and Mask R-CNNs with ease. Nevertheless, it is important to try building a model at least once from scratch to understand the math behind it.

**Detectron2** comes to the rescue if we want to train an object detection model in a snap with a custom dataset. All the models present in the model zoo of the Detectron 2 library are pre-trained on COCO Dataset. We just need to fine-tune our custom dataset on the pre-trained model. Detectron 2 is a complete rewrite of the first Detectron which was released in the year 2018. The predecessor was written on Caffe2, a deep learning framework that is also backed by Facebook. Both the Caffe2 and Detectron are now deprecated. Caffe2 is now a part of PyTorch and the successor, Detectron 2 is completely written on PyTorch. There are the various types of Object Detection models that the Detectron 2 offers. Using Detectron 2, Object Detection can be performed on any custom dataset using seven steps.

Step 1: Installing Detectron 2

Kickstart with installing a few dependencies such as Torch Vision and COCO API and check whether CUDAis available. CUDA helps in keeping track of the currently selected GPU and then install Detectron2.

Step 2: Prepare and Register the Dataset

Import a few necessary packages. Datasets that have builtin support in detectron2 are listed in builtin datasets. If you want to use a custom dataset while also reusing detectron2’s data loaders, you will need to **Register**your dataset (i.e., tell detectron2 how to obtain your dataset). There are certain formats in how data can be fed to a model such as a YOLO format, PASCAL VOC format, COCO format, etc. Detectron2 accepts the COCO Format of the dataset. COCO format of the dataset consists of a JSON file which includes all the details of an image such as size, annotations (i.e., bounding box coordinates), labels corresponding to it’s bounding box, etc.

Step 3: Visualize the Training Set

We’ll randomly pick 3 pictures from the train folder of our dataset and see how the bounding boxes look like.

Step 4: Training the Model

This is the step where we give configurations and set the model ready to get trained. Technically, we just fine-tune our model on the dataset as the model is already pre-trained on COCO Dataset. There are a ton of models available for object detection in the Detectron2’s Model Zoo. Here, we use the **faster\_rcnn\_R\_101\_FPN\_3x**model which looks in this way on a high level.

Step 5: Inference using the Trained Model

It’s time to infer the results by testing the model on the Validation Set. An output folder gets saved in the local storage after successful completion of training in which the final weights are stored.

Step 6: Evaluation of the Trained Model

Usually, the model is evaluated following the COCO Standards of evaluation. Mean Average Precision (mAP) is used to evaluate the performance of the model.

TOOLS AND TECHNOLOGY

**R-CNN**

It is used to bypass the problem of selecting a huge number of regions. It uses selective search to extract just 2000 regions from the image named as region proposals. Therefore, now, instead of trying to classify a huge number of regions, it just work with 2000 regions. These 2000 region proposals are generated using the selective search algorithm. In addition to predicting the presence of an object within the region proposals, the algorithm also predicts four values which are offset values to increase the precision of the bounding box.

**Faster-RCNN**

The approach is similar to the R-CNN algorithm. But, instead of feeding the region proposals to the CNN, we feed the input image to the CNN to generate a convolutional feature map. From the convolutional feature map, we identify the region of proposals and warp them into squares and by using a RoI pooling layer we reshape them into a fixed size so that it can be fed into a fully connected layer. From the RoI feature vector, we use a softmax layer to predict the class of the proposed region and also the offset values for the bounding box.

**OCR**

OCR systems transform a two-dimensional image of text, that could contain machine printed or handwritten text from its image representation into machine-readable text. Optical Character Recognition remains a challenging problem when text occurs in unconstrained environments, like natural scenes, due to geometrical distortions, complex backgrounds, and diverse fonts. OCR as a process generally consists of several sub-processes to perform as accurately as possible. The subprocesses are Preprocessing of the Image, Text Localization, Character Segmentation, Character Recognition, Post Processing.

**TextVQA**

TextVQA requires reading and understanding text in images to answer a question. TextVQA is based on custom pairwise fusion mechanisms between a pair of two modalities and are restricted to a single prediction step by casting TextVQA as a classification task.

**T5**

T5 is used to reframe all NLP tasks into a unified text-to-text-format where the input and output are always text strings, in contrast to BERT-style models that can only output either a class label or a span of the input. Text-to-text framework allows us to use the same model, loss function, and hyperparameters on any NLP task, including machine translation, document summarization, question answering, and classification tasks (e.g., sentiment analysis). We can even apply T5 to regression tasks by training it to predict the string representation of a number instead of the number itself.

**BART**

BART is a denoising autoencoder for pretraining sequence-to-sequence models. BART is trained by corrupting text with an arbitrary noising function, and learning a model to reconstruct the original text. It uses a standard Tranformer-based neural machine translation architecture . It evaluate a number of noising approaches, finding the best performance by both randomly shuffling the order of the original sentences and using a novel in-filling scheme, where spans of text are replaced with a single mask token.

**M4C**

Multimodal Multi-Copy Mesh is a approach for the TextVQA task based on a pointer-augmented multimodal transformer architecture with iterative answer prediction. Given a question and an image as inputs, we extract feature representations from three modalities – the question, the visual objects in the image, and the text present in the image. These three modalities are represented respectively as a list of question words features, a list of visual object features from an off-the-shelf object detector, and a list of OCR token features based on an external OCR system. Model projects the feature representations of entities (question words, detected objects, and detected OCR tokens) from the three modalities as vectors in a learned common embedding space. Then, a multi-layer transformer is applied on the list of all projected features, enriching their representations with intra- and intermodality context. Model learns to predict the answer through iterative decoding accompanied by a dynamic pointer network. During decoding, it feeds in the previous output to predict the next answer component in an autoregressive manner. At each step, it either copies an OCR token from the image, or selects a word from its fixed answer vocabulary