CS-725
Text Detection and Recognition from Images

Arnab Das	20305R005
Kunal Verma	203050121
Mallela Niteesh Kumar Reddy	203050065
Nimesh Aggarwal	203050049

Contents

1 Introduction

Text extraction from images is an important task that is helpful in uncountable number of scenarios from information extraction, text synthesis and various computer vision related tasks. Most of the information on the internet is in the form of images and videos so it becomes even more crucial to have a system that is efficiently able to extract the information such as texts from these images which can be further used in other task or for deriving inferences from the text. Here we have tried to explore the text recognition task by dividing it into smaller parts of text detection and text recognition and presented our analysis on these tasks.

2 Pipeline

The project has been divided into two major parts

1. Text Detection: getting the bounding boxes in image
   This part focuses on taking the images as input and outputting the coordinates of the region where text is present in the image. The regions would contain a single word.

2. Text Recognition: extracting words from the bounding boxes
   In this part we would extract the words from the regions that were identified by the text detection module. The regions would contain a single word and this module would extract the word from that region.

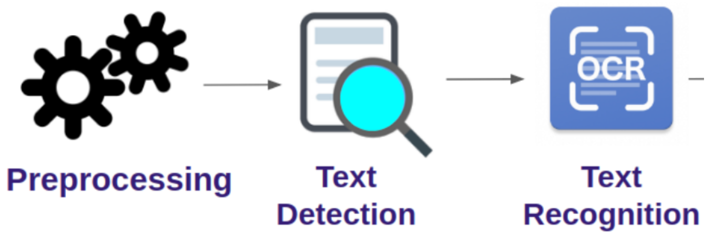


Figure 1: Pipeline for text extraction,

For each of these parts we have implemented several models and pipelined the combination of these models in order to generate the final text from the images. We have reported accuracy measures for each combination of text detection and recognition modules in the results section

3 Text Detection

3.1 YOLO(You only look once)

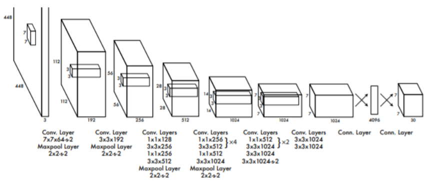
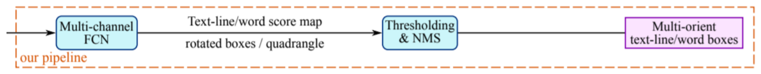


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

Our system divides the input image into an S × S grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object.Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and also how accurate it thinks the box is that it predicts.Each bounding box consists of 5 predictions: x, y, w, h, and confidence. The (x, y) coordinates represent the center of the box relative to the bounds of the grid cell. The width and height are predicted relative to the whole image. Finally the confidence prediction represents the IOU between the predicted box and any ground truth box.Each grid cell also predicts C conditional class probabilities, Pr(Classi —Object). These probabilities are conditioned on the grid cell containing an object.At test time we multiply the conditional class probabilities and the individual box confidence predictions, which gives us class-specific confidence scores for each box. These scores encode both the probability of that class appearing in the box and how well the predicted box fits the object.

3.2 EAST(Efficient accurate scene text detector)



The key component of the proposed algorithm is a neural network model, which is trained to directly predict the existence of text instances and their geometries from full images. The model is a fully-convolutional neural network adapted for text detection that outputs dense per-pixel predictions of words or text lines. This eliminates intermediate steps such as candidate proposal, text region formation and word partition. The post-processing steps only include thresholding and NMS on predicted geometric shapes.One of the predicted channels is a score map whose pixel values are in the range of [0, 1]. The remaining channels represent geometries that encloses the word from the view of each pixel. The score stands for the confidence of the geometry shape predicted at the same location

4 Text Recognition

4.1 CRNN + CTC model

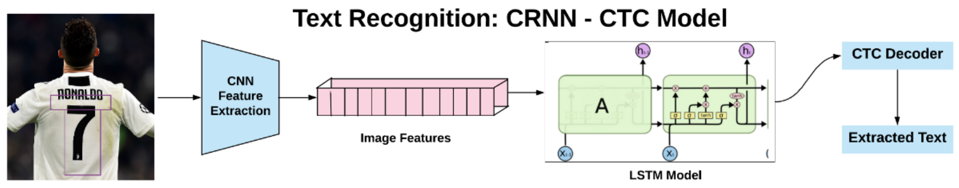


Figure 2: Text Recognition model

After the text detection part, the areas where the words were present is cropped and sent to the convolutional neural network feature extraction layers these features are fed to many-to-many LSTM architecture, which outputs softmax probabilities over the vocabulary.

4.1.1 CRNN Architecture

A grayscale image with width 128 and height 32 is sent through a series of convolution max-pooling layers.Layers are designed in such a manner that we obtain feature maps of the shape (None, 1, 31, 512) and reshaped to (None, 31, 512). Here None is batch size and 31 is time steps, 512 is number of features in every time step.Later these feature maps are fed to the LSTM model for every time step i.e., 31, we get a softmax probability over vocabulary.These outputs from different time steps are fed to CTC decoder to finally get the raw text from images.

4.1.2 Advantage of CRNN over CNN

As Sequence model like LSTM can predict the next sequence for every time step we can recognize missing or very hazy part of the image also depending on the context.For example if the images was Thank you but 'n' was not recognized from CNN features due to n being missing, we still can predict 'n' using LSTM

4.1.3 CTC Decoding

As length of the ground truth and length of time steps (predictions) are not same we are using CTC loss as now we need not worry about alignment details to compute loss.Every alignment in CTC is of same length as input which can have special blank symbol.To get the probability of an output given an input, CTC works by summing over the probability of all possible alignments which generates the ground truth after merging same characters and removing blanks.

4.1.4 CTC Loss

CTC loss = - log(Probability of getting ground truth), which sums over all possible alignments.CTC loss can be very expensive to compute if compute the score for each alignment summing them all up as we go.Thankfully, we can compute the loss much faster with a dynamic programming algorithm. The key insight is that if two alignments have reached the same output at the same step, then we can merge them.

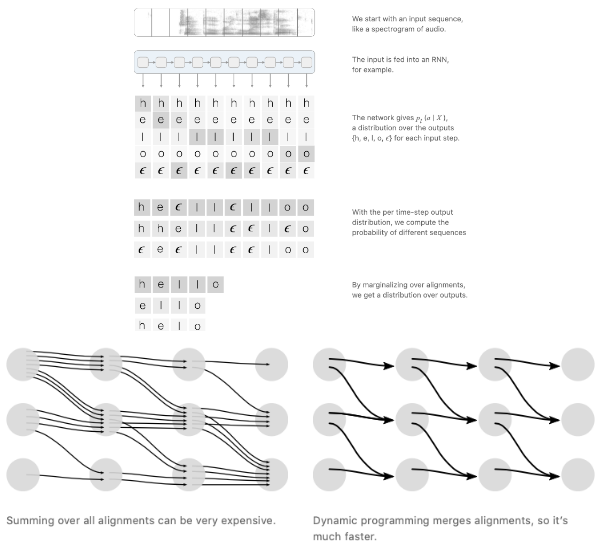

We start with an input sequence, like a spectrogram of audio.
The input is fed into an RNN, for example.
The network gives $p_t(a \mid X)$, a distribution over the outputs {h, e, l, o, ϵ} for each input step.
With the per time-step output distribution, we compute the probability of different sequences
By marginalizing over alignments, we get a distribution over outputs.
Summing over all alignments can be very expensive.
Dynamic programming merges alignments, so it's much faster.

Figure 3: CTC working mechanism and Use of dynamic progaming in CTC computation

5 Dataset

- Text Detection:
  - ICDAR 2017 dataset
  - 428 real images with annotations
- Text Recognition:
  - Synth90k dataset
  - 9 million images covering 90k English words
  - Used a small subset for training (52k images)

6 Results

We ran the pipeline with 2 text detection models(YOLO and EAST) with 2 different text recognition model(Self trained CRNN model on 52,000 images and another state-of-the-art CRNN model, i.e., KerasOCR). This gives us a total 4 combinations of methods and we have reported accuracy measures for each of these combinations.
For each combination we have reported

- Word-level accuracy: accuracy of of getting exact word from the image
- Average edit distance: How many characters differ between the actual and predicted word
- Character level precision, recall and f1-score

6.1 YOLO + CRNN(self trained)

1. Word Level Accuracy - 28.925%
2. Average edit distance - 1.87
3. Character Level
    - Precision - 0.84
    - Recall - 0.89
    - F1 Score - 0.86

6.2 EAST + CRNN(self trained)

1. Word Level Accuracy - 32.989%
2. Average edit distance - 1.77
3. Character Level
    - Precision - 0.85
    - Recall - 0.87
    - F1 Score - 0.85

6.3 YOLO + CRNN(KerasOCR)

1. Word Level Accuracy - 68.5%
2. Average edit distance - 0.77
3. Character Level
    - Precision - 0.93
    - Recall - 0.93
    - F1 Score - 0.92

6.4 EAST + CRNN(KerasOCR)

1. Word Level Accuracy - 56.7%
2. Average edit distance - 0.98
3. Character Level
    - Precision - 0.92
    - Recall - 0.94
    - F1 Score - 0.93

7 Analysis

7.1 Analysis on results

We can see that word level accuracy for self trained CRNN models with both text detection modules is low. However on a closer inspection we can see that the average edit distance between the actual and the predicted words is less than 2. It means on an average the predicted word differs only 2 characters from the actual word. On going to the character level we can see that the f1-score is quite good reaching upto 0.9.
In the self-trained crnn model, the images were also trained on the cropped images from YOLO model and EAST model separately. In the YOLO model there were instances of borders of bounding boxes coming inside the images. Those lines were interpreted as "l" character most of the time. This could be one of the reasons why the word level accuracy. On the other hand the EAST model did not have this problem so the word level accuracy came out to be more.
The character level precision for self trained CRNN model is almost same in both the cases

Coming to the KerasOCR model which was pre trained on 9 million images from Mjsynth dataset, the word level accuracy came far better than the self-trained model. This is something one can expect from the state-of-the-art models. Here the YOLO model + CRNN KerasOCR model performed better than the EAST model as the state-of-the-art model was not trained on the images which contained these bounding boxes. The character level accuracies also reached upto 0.95 in these cases. With an average edit distance of less than 1, the words were most of the times predicted fairly accurately. The character level confusion matrix and the character level precision, recall and f1-scores suggests that individual character recognition is quite good which can be attributed the Bilstm layers present in CRNN network which treats image as a sequential data using the concept of receptive fields.

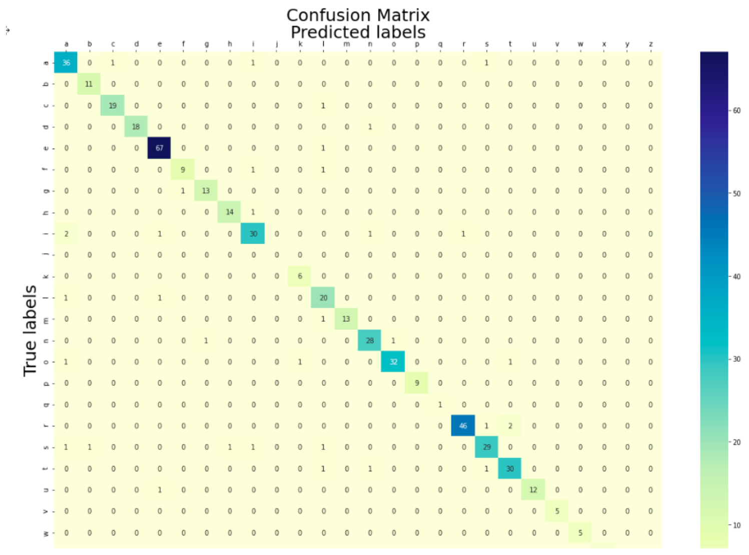


Figure 4: Confusion matrix - YOLO + KerasOCR

8 Github link

Github Repository which contains full Source codehttps://nimesh@git.cse.iitb.ac.in/nimesh/FML.git

9 References

1. https://github.com/Neerajj9/Text-Detection-using-Yolo-Algorithm-in-keras-tensorflow/blob/master/Yolo.ipynb

2. https://github.com/kaish114/Real-Time-Text-Detection-Using-YOLOV2/blob/master/Real

3. https://wandb.ai/authors/text-recognition-crnn-ctc/reports/Text-Recognition-With-CRNN-CTC-Network–VmlldzoxNTI5NDI

4. https://github.com/rajesh-bhat/spark-ai-summit-2020-text-extraction

5. https://github.com/JaidedAI/EasyOCR

6. https://github.com/faustomorales/keras-ocr