DATA 2040

# Optical Character Recognition Based on Single Shot MultiBox Detector and Convolutional Recurrent Neural Network

**Enmin Zhou**
Data Science Initiative
Brown University
enmin_zhou@brown.edu

**Huaqi Nie**
Data Science Initiative
Brown University
huaqi_nie@brown.edu

**Haoda Song**
Data Science Initiative
Brown University
haoda_song@brown.edu

**Siyuan Li**
Data Science Initiative
Brown University
siyuan_li2@brown.edu

**Yangyin Ke**
Data Science Initiative
Brown University
yangyin_ke@brown.edu

## Abstract

With a wide variety of applications in both academia and industry, Optical Character Recognition deals with the problem of recognizing all kinds of different characters from images. In this paper, we construct a model pipeline that combined the technology from Single Shot MultiBox Detector and Convolutional Recurrent Neural Network to obtain the text information in natural street views. While our recognized text result lacks the robustness for detecting certain unique-style fonts, we show that this implementation is a promising methodology to recognize text from various natural backgrounds, and would benefit from further exploration in future works.

## I. Introduction

Optical Character Recognition (OCR) has long been a challenging computer vision and deep learning problem. The basic steps of Optical Character Recognition (OCR) consist of three major steps, Text Detection, Text Recognition, and Post-Processing. As Text Recognition in a natural environment become more sophisticated because of frequent blurs, distortions, and clutters, in this paper, we hope to construct a model that takes advantages of convolutional neural networks (CNNs), recurrent neural networks (RNNs) and the architecture of Single Shot MultiBox Detector (SSD) to extract characters.

In Text Detection, SSD outperformed the traditional models such as Faster R-CNN and YOLO. It uses VGG-16 model pre-trained on ImageNet as the base model and adds multiple convolutional layers of decreasing sizes as the top. VGG-16 extracts the useful image features from images and then object detection happens in each convolutional layer. The decreasing sizes prevent the objects with various sizes, either large or small, could be captured well.

In Text Recognition, Convolutional Recurrent Neural Network (CRNN) is a novel neural network architecture that integrates the advantages of both Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). As the text recognition task has long been performed in deep learning study, CRNN model can take input images of different dimensions and return the results with different lengths. We can directly put coarse text input in the training process without detailed annotations. CRNN does not utilize fully connected layers that are widely applied in CNN. This act helps the model be a much more compact and efficient model.

We outline our pipeline OCR architecture in Figure 1.

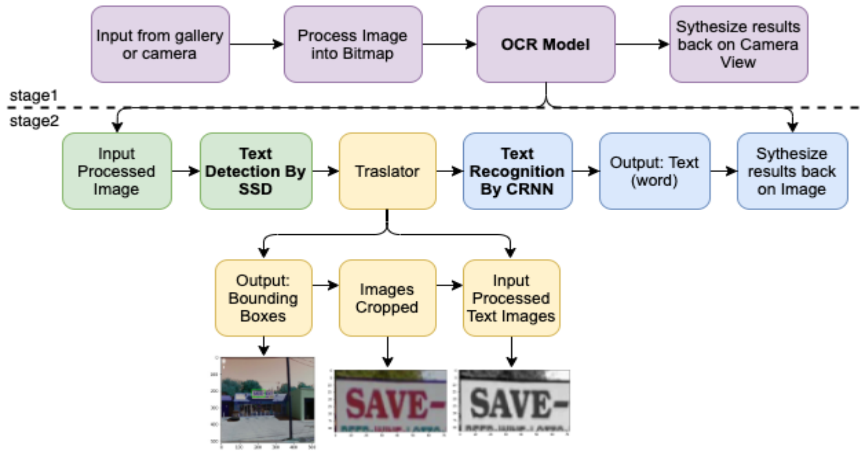

*Figure 1 Pipeline Architecture*

## II. Data

We use Google IIIT5K to train the text recognition model and fine-tune the pre-trained text detection model on Google Street View dataset. Google IIIT5K data contains 5000 cropped word images from scene texts and born-digital images. The pre-trained SSD text detection model is trained on the ICDAR competition dataset and then we fine-tune the model on Google Street View dataset (STV). SVT contains approximately 400 ground-truth scene text bounding boxes from 100 natural street view images harvested from Google Map. To simplify the process of YOLO loss computation, we calculate the upper left corner and lower right corner of the bounding box with original height and width, which are then further converted into YOLO matrix. From this preprocessed set, we split our data into train and validation so that approximately 300 bounding boxes with 75 images are in the training set and 80 bounding boxes with 25 images are in the validation set.

## III. Methods

### 3.1 Single Shot MultiBox Detector (SSD)

The text detection model follows the method of Single Shot Multibox Detector. For input shape, we choose 512*512 instead of 224*224, which would keep the text resolution higher for feature extraction. For architecture, we integrate the pretrained MobileNet V2 as a feature extractor, remove a few fully connected layers on top and add 3 more convolutional layers with a reshape layer for bounding box output. The MobileNet V2 is chosen for its high accuracy and less number of weights, which is designed towards a mobile application. The original fully connected layers are removed because we do not need classification predictions, but bounding box predictions. Therefore for the output shape , the new output shape is reshaped into a vector of 4 dimensions (grid height, grid width, 1, 5) for YOLO loss computation.

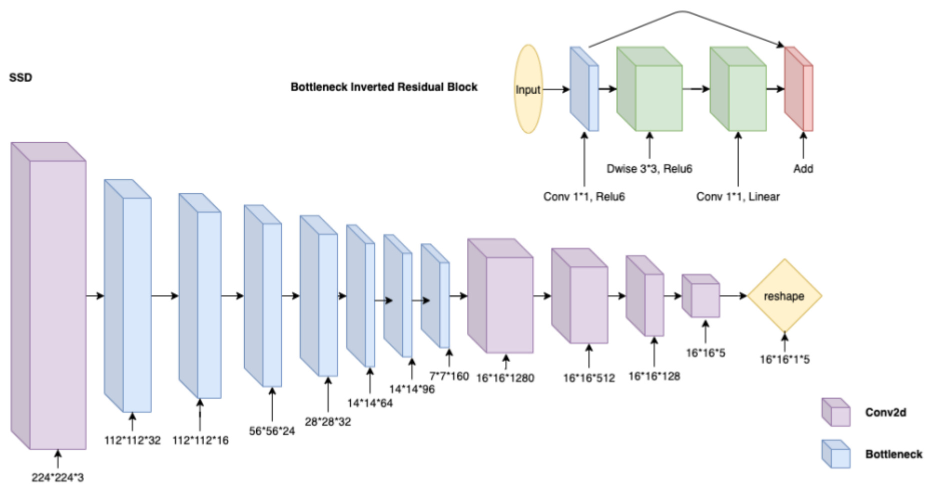

*Figure 2 SSD Model Architecture*

- **YOLO loss**

Yolo algorithm is based on IOU calculation with respect to different aspects of the ground truth. In our model, the total loss is the sum of localization error and classification error. The localization error is calculated in two dimensions: the first one is the sum of the square of Manhattan distance of x anchor and y anchor between output and label for each bounding box; the second one sum-squared loss of the predicted height and width (The predictions are in squared form to alleviate the bias between large and small bounding boxes). Both parts are multiplied by term ƛcoord to control the loss of this part to control its weights on total loss.

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[ \left(x_i - \hat{x}_i\right)^2 + \left(y_i - \hat{y}_i\right)^2 \right]$$
$$+\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[ \left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 \right]$$

The Classification error is special in our case, since we only have 1 class in our dataset, which is the text object. Therefore, the classification error is the sum of the absence error and presence error. For presence error which contains ground truth objects in prediction, it is the sum-squared error between the predicted confidence score and the ground truth for each bounding box in each cell. For absence error which does not contain an object in prediction, sum-squared error of the cells which do not contain any objects. Additionally, we add a term of ƛnoobj to control the loss of this part since we do not punish too much on absence, but we do care about the presence part.

$$+\sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i\right)^2$$
$$+\lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i\right)^2$$

In the end, we fine-tune our SSD model on the preprocessed SVT data to increase its capacity on detecting long-distance text objects.

## 3.2 Convolutional Recurrent Neural Network (CRNN)

To establish our novel CRNN model, we include the convolutional layers and bi-directional LSTM layers into the model architecture and use the reshape layer to combine these two blocks.

In the CNN part, we would normalize each batch and followed by ReLU activation the drop-out layer to prevent gradient vanishing and overfitting.  To reduce the noises in the images, we consider the auto-encoder when construct the dimensions of each layer. Combined with the max-pooling layers, this method could help the model capture the main features in the images. The outputs of CNN blocks would be reshaped then passed into the RNN part. We use bi-directional LSTM to try to improve the accuracy of the character prediction in the images. The final outputs are passed through the softmax activation to classify the images into character- level result with different lengths.

For the model training, we utilize the following callbacks mechanisms. The early stopping would stop the training when the validation loss metric has stopped decreasing for 10 epochs, and the checkpoint function could help to save the current best model in the training process, and the reduce learning rate function could adjust the learning rate after each epoch. Then, the model is trained with Adam optimizer and CTC loss.

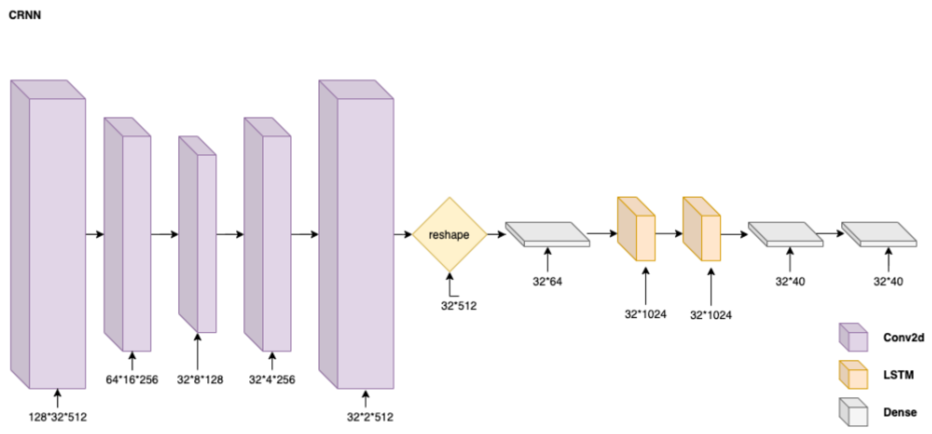

*Figure 3 CRNN Model Architecture*

## IV. Results and Discussion

- **Text detection model (SSD MobileNet V2):**

Compared with the pre-trained model, our fine-tuned model does make some improvements. After fine-tuning the pre-trained model with our street view dataset, we get a model with optimized weights that works better to distinguish between text and non-text bounding boxes. Additionally, our fine-tuned model is more sensitive to large texts than the pre-trained model. However, improvements are still needed regarding large-text sensitivity. In terms of recognizing blurred and tiny texts, our model is performing well, capable of predicting clear bounding boxes for distant texts.

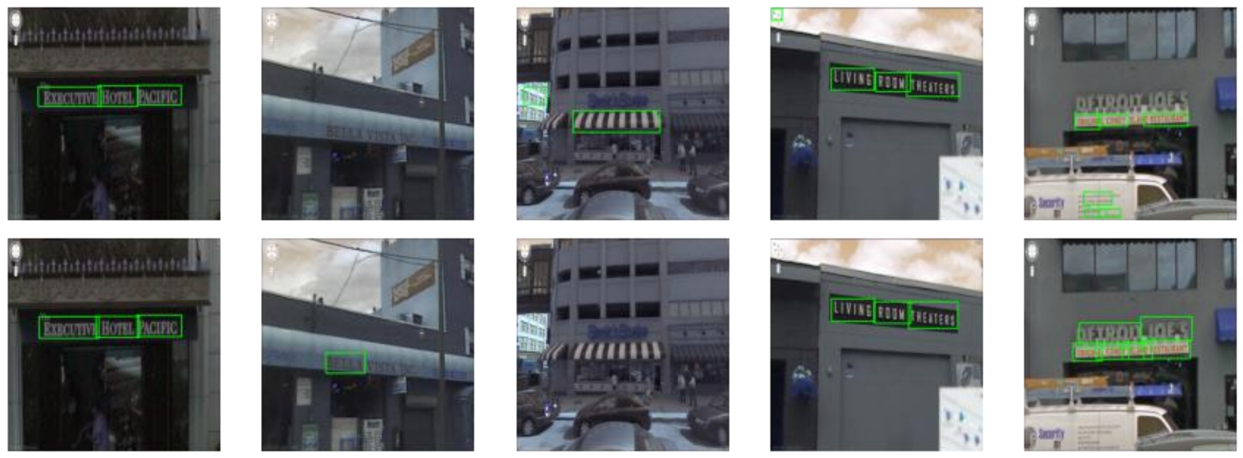
*Figure 4 Bounding boxes predicted by pre-trained model V.S fine-tuned model*

- **Text Recognition model (CRNN):**

| Model | Character-level Accuracy | Word-level Accuracy |
|---|---|---|
| Baseline (VGGNet+RNN) | 39.61% | 29.60% |
| CRNN (Fine-tuned) | 68.78% | 50.20% |

For our final CRNN text recognition model, we trained 4000 images from the Google IIIT5K dataset and selected 500 images for the validation set. To understand our text recognition model in advanced perspective, we calculated the accuracy score for the prediction result based on two different levels. One is the character level, and the other is the word level. We summarize the model improvement from the baseline to the find-tuned in the above table. We can see that our model has a satisfying detection accuracy given the limited resource of text image data.

- **Overall result**

Combining the text detection model and the text recognition model, we are able to read the texts in images. The complete pipeline performs well in most cases, with accurate bounding boxes and text recognition. We list sample recognition results below to demonstrate the overall model performance.

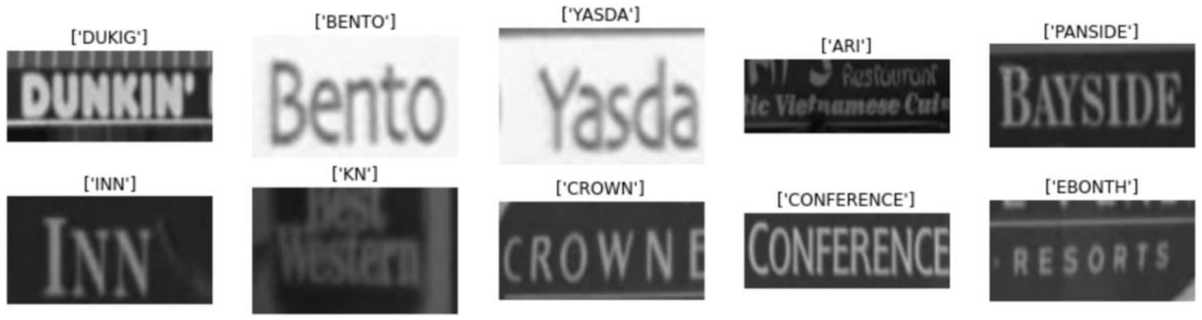

*Figure 5 Final Text Recognition Result*

## V. Conclusion and Future Work

We conclude our model detects and recognizes the natural scene text from street view accurately, especially for small text which is far away from the photographers. Additionally, the complete pipeline performs well on recognizing artistic text. However, in a sense, if bounding boxes can provide closer boundary lines, avoiding containing letters of the next word, the text recognition model will be given a more concise input to better recognize corresponding words. Therefore, in order to balance the tradeoff between the small and large sizes of text from the natural images, our model loses some ability to detect and recognize the extremely large text object. Besides the lack of extremely large text detection, the outputs of the model are unable to recognize the special characters.
However, one of the potential improvements of the model could be using larger dataset to train the model so that the model would capture more styles and patterns of text. Post OCR correction would also enhance the accuracy of the predictions, such as the correction of the output using BERT but this method may require a large lexicon.

Furthermore, the OCR application could be set up on the Android platform with SDK 28 and NDK bundle 18r. The application depends on the external packages org.tensorflow - android and org.tensorflow - lite. For model, we put our trained model frozen graph in assets in the form of protobuf. For input, we can choose an image either from a gallery or camera. Assets are read as a stream and the application decodes the stream as bitmap for model input. The output from the model, which are bounding boxes and word predictions, will be embedded onto the image. Finally, the synthesized image is binded to the view of the main activity on the screen.

## VI. Reference

[1] Mishra, A. and Alahari, K. and Jawahar, C.~V. (2012). *Scene Text Recognition using Higher Order Language Priors* [Dataset]. http://cvit.iiit.ac.in/projects/SceneTextUnderstanding/IIIT5K.html

[2] Neeraj Panse. (2019). *Text-Detection-using-Yolo-Algorithm-in-keras-tensorflow* [Dataset]. https://github.com/Neerajj9/Text-Detection-using-Yolo-Algorithm-in-keras-

[3] Kai Wang, Boris Babenko and Serge Belongie. (2010). *Street View Text | UC San Diego* [Dataset]. http://vision.ucsd.edu/%7Ekai/svt/

[4] What is OCR? Introduction to Optical Character Recognition | Anyline. (n.d.). ANYLINE. https://anyline.com/news/what-is-ocr/

[5] Gandhi, R. (2018, December 3). *R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms*. Medium. https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e#:%7E:text=The%20reason%20%E2%80%9CFast%20R%2DCNN,map%20is%20generated%20from%20it

[6] Ren, S. (2015, June 4). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. ArXiv.Org. https://arxiv.org/abs/1506.01497

[7] Benabderrazak, J. (2020, April 9). *OpenCV EAST model and Tesseract for detection and recognition of text in natural scenes*. Medium. https://jaafarbenabderrazak-info.medium.com/opencv-east-model-and-tesseract-for-detection-and-recognition-of-text-in-natural-scene-1fa48335c4d1

[8] Shrimali, V. (2021, March 20). *Deep Learning based Text Detection Using OpenCV*. Learn OpenCV | OpenCV, PyTorch, Keras, Tensorflow Examples and Tutorials. https://learnopencv.com/deep-learning-based-text-detection-using-opencv-c-python/

[9] Liu, W. (2015, December 8). *SSD: Single Shot MultiBox Detector*. ArXiv.Org. https://arxiv.org/abs/1512.02325

[10] Weng, L. (2018, December 27). *Object Detection Part 4: Fast Detection Models*. Lil'Log. https://lilianweng.github.io/lil-log/2018/12/27/object-detection-part-4.html#ssd-single-shot-multibox-detector

[11] Liao, M. (2018, January 9). *TextBoxes++: A Single-Shot Oriented Scene Text Detector*. ArXiv.Org. https://arxiv.org/abs/1801.02765

[12] *Zero to Hero: Guide to Object Detection using Deep Learning: Faster R-CNN,YOLO,SSD*. (2017, December 28). CV-Tricks.Com. https://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/

[13] E. (2020a). *emedvedev/attention-ocr*. GitHub. https://github.com/emedvedev/attention-ocr

[14] S. (2020b, August 25). *Handwriting Recognition using CRNN in Keras*. Kaggle. https://www.kaggle.com/samfc10/handwriting-recognition-using-crnn-in-keras

[15] M. (2019). *MaybeShewill-CV/CRNN_Tensorflow*. GitHub. https://github.com/MaybeShewill-CV/CRNN_Tensorflow

[16] *Use a TensorFlow Lite model for inference with ML Kit on Android*. (2021). Firebase. https://firebase.google.com/docs/ml-kit/android/use-custom-models

[17] *Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen*; MobileNetV2: Inverted Residuals and Linear Bottlenecks; Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 4510-4520