POLITECNICO DI TORINO

Department of Electronics and Telecommunications
Master of Science in ICT for Smart Societies

Master Degree Thesis

Optical Character Recognition

Supervisor:
Prof. Monica Visintin

Author:
Ani DEVER

April 2018

DEDICATION

I dedicate this dissertation work to my inspiring parents, for their endless support and encouragement during the challenges of my life.

ACKNOWLEDGEMENTS

I wish to present my special thanks to Autour team of Shared Reality Lab, McGill University. They have been a great team, whose help and motivation made my research successful.

I would like to express my gratefulness to Mathieu Bouchard, the lead developer of Autour, who taught me a lot about computer science and mobile application development.

I would like to pay my regards to Professor Monica Visintin, my advisor at Politecnico di Torino, who inspired my interest to machine learning and deep learning. Her encouragement made it possible to achieve my goals.

I would like to show my warm thank to Professor Jeremy Cooperstock, my advisor at McGill University, who gave me the chance being his intern and explore the research activities at his lab. I appreciate his trust in me, his moral and financial support.

ABSTRACT

Autour[5] [1] is an eyes-free, cartographical mobile application being developed by Shared Reality Laboratory of McGill University as a research project. Autour presently runs on iPhones starting from iPhone 4S and the corresponding Android version is being developed. The application is designed to give visually impaired individuals a better sense of their surroundings. It can be used hands-free by leaving the phone in a bag around the user's neck or handheld. It utilizes inbuilt capabilities and sensors of the smart phone such as accelerometer, gyroscope, compass and Global Positioning System (GPS), to determine the user's location and orientation. Then the relevant sound instructions are presented to the user either through a bone-conducting or open air headphones so as not to interfere with the ambient sounds. The application's aim is to use ambient audio to expose the sort of information that visual cues, e.g signs and markings, give to individuals with sight.

Autour has several modes: the tilt of the device allows users to choose between two of the modes: horizontal mode and vertical mode. By default, the vertical mode is Radar mode and the horizontal mode is Beam mode. In total, user has following modes: Shockwave, Browse, Tutorial, Menu and a mode that just waits for GPS lock. In any horizontal and vertical mode, user can tap the screen twice quickly to hear the address where the user is located, the status of the sensors and a summary of places around the user. While that is spoken, user may interrupt by tapping the screen once or tilt the device.

There are two different Sweep modes: Radar and Shockwave. Sweep

[1] http://autour.mcgill.ca/en/

modes are used when the device is being used horizontally. The selection of the sweep modes can be done in the settings. In Radar mode, a tap will start an automatic sweep when the device is pointed upwards with the screen facing towards the user. The user will hear a ticking sound indicating the progress of the scan, along with the names of the places around the user sorted by distance. In Shockwave mode sorting is done by direction, therefore the sweep is done as a circle starting from where the user is and grows until the specified maximum distance is reached.

Beam mode can be activated and deactivated in the same manner as in Sweep modes. When the Beam mode is activated the user will hear a tick, then places that are in that sector are enumerated in order of increasing distance.

Browse mode creates a list of places nearby and reads it to the user. The user can navigate through the list by swiping. Standby mode starts automatically when GPS does not work well. Autour stays in Standby mode until GPS data is available again.

Autour is a mobile application depending on external services such as GPS, Foursquare™, Google Places™ or OpenStreetMap which can be unreliable. The application may fail if certain conditions are not satisfied e.g. if the user is traveling in a vehicle or if one the external services break unexpectedly. Autour does not recommends its user to rely on it for navigation and safety, it is not designed for mobilized user. It is not a replacement for a cane or a guide dog.

Shared Reality Lab is hosting the Autour team composed of a lead developer and students with different levels and backgrounds, a team of individuals with different interests and skills. Current objective is to add new features to Autour (such as scene description, chat boxes and optical character recognition) in order to provide a broad visual feedback to its users. Ideally, the users will be able to know what kind of scene they are in, what is written in their point of view and will be able to ask questions about what is happening around.

The visual task of classifying an object once was a big challenge. Yet, recent advancements in artificial intelligence have sparked major progression in computer vision. Now machines are drastically successful at computer vision problems such as pattern recognition, object detection and classification. Currently, the most accomplished methods in the literature are deep learning based. Today it is possible build strong computer vision systems by the help of deep learning algorithms and frameworks.

This thesis documents the effort given to assemble an optical character recognition (scene text detection and recognition to be precise) system by gathering state-of-the-art methods. The primary goal of this thesis work is to build a system by pipelining two different open source projects provided by the authors & developers of the methods [69, 51]. Ultimately Autour will be able to read the texts out loud, present in the user's scene.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
Introduction

According to World Health Organization (WHO) there are approximately 253 million people living with visual impairment: 36 million are blind and 217 million have moderate to severe vision impairment as of September 2017. Of these visually impaired people 90% are living in low- and middle-income countries. Furthermore WHO estimates that up to 80% of visual impairment and blindness in adults is preventable or treatable. Important progress is already being made by international communities such as WHO and World Blind Union in order to fight avoidable blindness. Besides that, the existence of projects like Autour is remarkably improving the life quality of blind community.

Being able to read visible texts in natural scenes e.g name tags at a door, street nameplates, store names, has a considerable importance in our daily life as they convey essential information about the environment. A technology allowing access to such information could provide a better independent travel experience, environmental awareness and improved self-confidence to blind users. This is the exact purpose of this thesis work, adding an end-to-end scene text detection and recognition engine to Autour in favor of exposing such opportunity.

Optical Character Recognition, abbreviated as OCR, can be defined as the identification and conversion of printed, typed or handwritten characters into machine encoded text. It is a process of transforming and importing already existing texts into machine environment. On the other hand, scene text detection and recognition is an open and more complicated task when compared to standard OCR. Scene texts do not necessarily have uniform background or text alignment, they might appear in random sections of the image in different fonts and languages. These facts makes classic OCR techniques inadequate in Autour context. An example of scene text detection can be seen in Fig 1–1.

Figure 1–1: Scene text detection
figure taken from ICDAR's website: http://rrc.cvc.uab.es/?ch=2&com=tasks

CHAPTER 2
Background

2.1 OCR history

The idea of machines imitating human abilities is a long coming idea, and character recognition has always been a subset of this idea. The invention of character recognition is considered as Charles R. Carey's retina scanner. Later on in 1954 The Reader's Digest magazine installed an OCR mechanism to convert the sales reports into punched cards. As these advancements have triggered the progression, OCR started to have a bigger role in our life as in form of postal, passport and price tag scanner.

Nowadays, OCR is a domain of computer vision which obtains increasing popularity and significance. Top level conferences such as Conference on Computer Vision and Pattern Recognition (CVPR) and International Conference on Computer Vision (ICCV) have witnessed major discoveries in computer vision and pattern recognition. The International Conference on Document Analysis and Recognition (ICDAR) is a leading conference touching to character and text recognition along with camera and video based scene text analysis. The ICDAR hosts number of competitions: Arabic, Chinese Handwriting Recognition Competition, Writer Identification Contest. Nonetheless the most promising one for Autour's case is the Robust Reading Competition. "Robust Reading"[1] refers to the research area dealing with the interpretation of written communication in unconstrained settings such as born-digital (such as the

[1] http://rrc.cvc.uab.es/

ones used in Web pages and email messages) and real scene images and videos. Typically Robust Reading is linked to the detection and recognition of textual information in scene images. The competition is organized around challenges that represent specific application domains for robust reading. Challenges are selected to cover a wide range of real-world situations such as Incidental Scene Text, Focused Scene Text [27].

2.2 Methodologies

Complete text detection and recognition systems can be analyzed with two commonly used methodologies: stepwise (see Fig 2-1) and integrated (see Fig 2-2). Integrated methodologies, have a goal of recognizing words where the detection and recognition procedures share information with character classification and/or use joint optimization strategies [64]. With an integrated methodology, character classification responses are considered the primary cues, and shared with detection and recognition modules. Stepwise methodologies, by contrast, have separated detection and recognition modules, and use a feed-forward pipeline to detect, segment and recognize text regions. They typically employ a coarse-to-fine strategy, which first localizes text candidates, and then verifies, segments, and recognizes them [61].

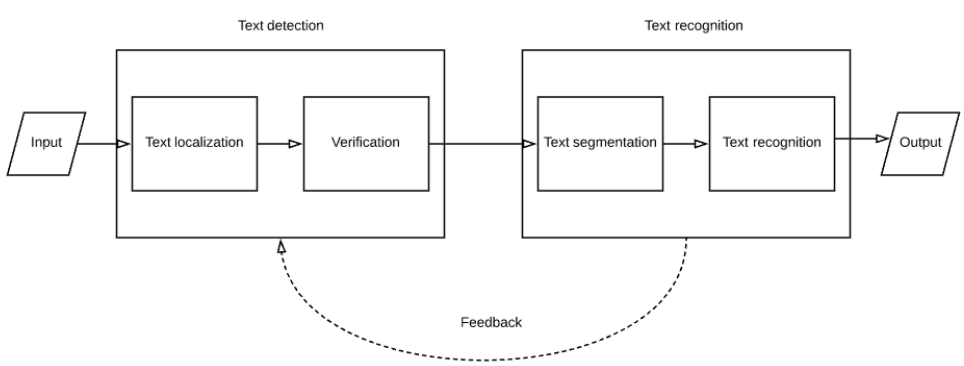

Figure 2–1: Stepwise methodology

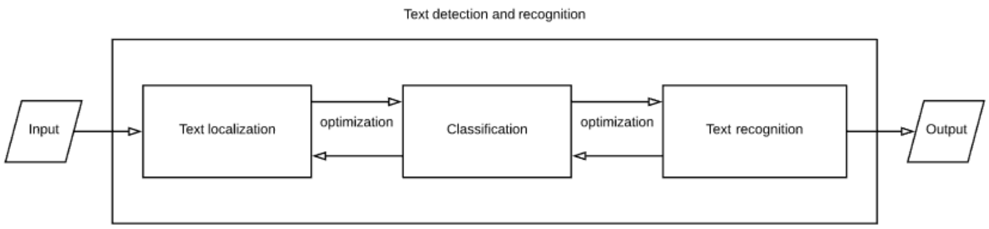

Figure 2–2: Integrated methodology

2.2.1 Text Detection

Existing methods for text detection can be roughly categorized into two major groups: connected component (CC) based methods [7, 11, 55], and region based methods (also called sliding window based methods) [4, 44, 8].

The region based methods use a multi-scale window to exhaustively scan through an image for candidate text regions. Then these regions are classified as text or non-text regions by a classifier which may exert intensity histogram, gradients or edges. Candidate text regions are first found with an edge map or gradient information. Subsequently, a refinement stage is conducted using heuristic rules or learned classifiers [66]. Region based methods commonly utilize an Adaboost classifier [31, 15, 8]. Recently deep learning approaches [59, 25, 9] are exploited, unsupervised learning techniques are used in [9] individually for detection and recognition, a linear SVM classifier is employed to classify the candidate regions. Although these methods can detect text effectively with high recall rate and robust to noise, their classification can be sensitive to false positives due to the large number of candidates. But due to heavy computations for intensive window scanning and advanced classification, these approaches are unsatisfactory to real-time applications.

The connected component(CC) based methods follows the connected component analysis theory, a case where subgroups of CCs are uniquely labeled hinge on heuristics. The candidate text regions are generated based on extracted CCs of the region, which may differ by the used properties e.g spatial layout, color [65], edge [22], texture [28] or gradient [32] features. Color features are used under the assumption of "text is often produced in a consistent and distinguishable color so that it contrasts with the background" [33]. The family of edge/gradient-based approaches assumes that text exhibits a strong and symmetric gradient against its background. Thus, those pixels with large and symmetric gradient values could be regarded as text components. [64]. Then non-text regions are eliminated, a conditional random fields (CRFs) model is adopted in [46] for this purpose whereas in [70], a neural network is used. The advantage of the connected component methods is that their complexity typically does not depend on the properties of the text (range of scales, orientations, fonts) and that they also provide a segmentation which can be exploited in the OCR step. Their disadvantage is a sensitivity to clutter and occlusions that change connected component structure [44].

Especially, Maximally Stable Extremal Regions(MSER) [39] and Stroke Width Transform(SWT) [13] are two breakthrough techniques which already have achieved impressive performance in scene text detection. MSER is an affine feature region detection algorithm used for blob detection in images. The MSER algorithm extracts a number of co-variant regions called MSER which are connected components of the appropriately thresholded image. A new set of elements alleged extremal regions are introduced, these elements are distinguished regions possessing important properties: they are closed under the affine transformation of image coordinates and invariant to affine transformation of intensity. The word 'extremal' refers to the property that all pixels inside the MSER have either higher (bright extremal regions) or lower (dark extremal regions) intensity than all the pixels on its outer boundary. The 'maximally stable' in MSER describes the property optimized in the threshold selection process [40]. These properties makes MSER robust against scale, view point and lighting changes. This is why MSER is a widely adopted algorithm in context of text detection among the other region detectors e.g Harris-affine and Hessian-affine. An example of MSER method is depicted in Fig 2–3.

Figure 2–3: MSER detection of a sample image from ICDAR

The use of MSER based methods for text detection and recognition started with [11], where MSERs are classified using cross-correlation with training templates. However, its sensitivity against blur makes it unsatisfactory to utilize without an auxiliary technique; in [7], MSER and Canny edges [6] are combined to obviate MSER's sensitivity against blur, which is achieved by removing the pixels outside the boundaries formed by Canny edges. In [23, 55] an intersection of Canny edges with MSER region is taken into consideration. Then a Stroke Width Transform step is placed after region detection by

Canny enhanced MSER. An example of MSER method is depicted in Fig 2–4 (OpenCV implementation of Canny [6]).

Figure 2–4: Canny edge detection of a sample image from ICDAR

Stroke Width Transform assumes that attached characters constructing text blocks are sharing similar attributes suchlike size, color and especially stroke width. SWT is a local image operator which computes per pixel the width of the most likely stroke containing the pixel. The output of the SWT is an image of size equal to the size of the input image where each element contains the width of the stroke associated with the pixel [13]. The biggest restraint of SWT method is its reliance on edge detection which may be unsuccessful in presence of blur and low-contrast. The approach defined in [23] uses SWT as filtering, text candidate regions where the stroke width varies largely are eliminated. Where [7] proposes a way of determining the stroke width based on distance transform, guaranteeing to provide stroke width information at every pixel of the original connected component with any stroke form. Different ways of enhancing SWT are introduced in [3] by edge orientation variance(EOV) and opposite edge pairs(OEP) or a combination of spatial-temporal analysis in [38].

However, these methods fall behind of those based on deep neural networks, in terms of both accuracy and adaptability, especially when dealing with challenging scenarios, such as low resolution and geometric distortion [69]. Deep learning based methods [68, 56, 35, 69, 26] achieved notable performance when compared to traditional methods. A Fully Convolutional Network (named Text-Block FCN) is used to generate a pixel-wise text/non-text salient map in [68], text blocks are detected via the FCN, followed by multi-line oriented text line extraction considering MSER components. The approach described in [56] introduces a novel Connectionist Text Proposal Network, a joint Convolutional Neural Network - Recurrent Neural Network model that directly localizes text sequences in convolutional layers. Authors of [56] have also developed a vertical anchor regression mechanism that jointly predicts vertical location and text/non-text score of each text proposal. The sequential proposals are naturally connected by a recurrent neural network, a Bi-directional Long Short Term Memory. Even though the method is reliable at horizontal and multi-scale text detection, it is not as competent as for text written at an angle. TextBoxes [35] is an end-to-end scene text detector inspired by SSD [36] and inherits the popular VGG-16 architecture [54]. Authors of [35] have proposed a single neural neural network to detect texts by directly predicting word bounding boxes. The approach adopts CRNN [51] as text recognizer in conjunction with TextBoxes, therefore it results in a simple pipeline and a single network to train. The work in [69] is named EAST, a simplistic scene text detection pipeline exploiting an FCN. The network has two stages only: an FCN which directly produces word or text-line level predictions, excluding redundant and slow intermediate steps such as candidate proposal, text region formation and word partition and NMS stage to obtain ultimate results. The method proposed in [26] is called Rotational Region

CNN based on Faster R-CNN [48] architecture. Authors take advantage of a Region Proposal Network to generate bounding boxes enclosing the candidate regions containing text. These proposals are classified and bounding boxes are refined before NMS is utilized to post-process the region candidates to yield the final results.

2.2.2 Text Recognition

Text recognition can be defined as the task of converting regions containing text into strings. Traditionally, text recognition has been focused on document images, where OCR techniques are well suited to digitize planar, paper-based documents. However, when applied to natural scene images, these document OCR techniques fail as they are tuned to the largely black-and-white, line-based environment of printed documents [25]. Despite the efforts and improvements, scene text recognition still remains as a challenging task considering the fact that the scene text regions might have complex background and non-uniform text patterns. In scene text detection-recognition case, detection modules generates bounding boxes around text candidate regions, after this point text recognition attempts to recognize the text represented within the box or refuse the box in case of a false positive detection.

The text recognition problem has been addressed in the literature on multiple levels: character recognition [9, 50], word recognition [45, 43] and text detection. The character recognition problem implicates constructing a recognizer to attain a probability distribution over all characters, when a character containing image is introduced. The character recognition problem is a classification problem that is generally addressed with the use of strong classifiers such as CNNs in [59], deformable parts models [53] or manually-engineered feature-extraction followed by a classifier [12]. The word recognition problem is, much like phone recognition and handwriting recognition, a sequence recognition problem. Previous works have addressed this problem using CNNs [60], Conditional Random Fields (CRFs) [43, 45] and Pictorial Structures (PS) [2]. As for scene text (cropped word) recognition, the existing methods can be grouped into segmentation-based word recognition and holistic word recognition. The main method for scene text recognition is considered as segmentation-based word recognition. In general, segmentation-based word recognition methods integrate character segmentation and character recognition with language priors using optimization techniques, such as Markov models and CRF [66]. Methods of text segmentation are:

- text binarization: operates to extract text pixels and remove the background pixels
- text line segmentation: attempts to convert a region of multiple text lines into multiple sub-regions of single text lines
- character segmentation: separates a text region into multiple regions of single characters [64]

An instance of text segmentation is shown in Fig 2–5.

Figure 2–5: Text segmentation
figure taken from ICDAR's website: http://rrc.cvc.uab.es/?ch=2&com=tasks

Text segmentation step is used by approaches [4, 25] to achieve accurately bounded characters. These approaches treat isolated character classification and subsequent word recognition separately. As they read each character independently they do not unleash the full potential of word context information in the recognition. Nonetheless their performance is severely harmed by the difficulty of character segmentation or separation. Importantly, recognizing each character independently discards meaningful context information of the words, significantly reducing its reliability and robustness [17]. Great portion of the work in this field relies on lexicon-dependent approaches. A lexicon is a set of label sequences that prediction is constrained to, *e.g.* a spell checking dictionary [51]. As it currently stands, it is difficult to recognize words with high accuracy without any language model due to the character confusion problem, therefore, all of the previous systems rely on lexicons to improve the results. However, since lexicons can be very large, authors of [2] make the distinction in our approach between where query time is linear in the size of the lexicon and those approaches where it is constant.

Given that texts are formed by sequentially ordered characters it is possible to see them as sequence-like objects. A system aspiring to recognize these objects are supposed to predict a series of labels, therefore this task can be cast as a sequence recognition problems where the length of the sequence may vary severely. RNNs are primarily designed to handle sequences and they are able to effectively learn continuous sequential features. Novel approaches [51, 17] are exploiting RNN's capability of learning continuous sequential features successfully by combining it with convolutional layers in order to leverage both the advantages of CNN and RNN.

CHAPTER 3
Design and Implementation

3.1 Design

As described and mentioned before, the goal of this dissertation is to design, build and integrate a scene text detector-recognizer to Autour for a possible use case that can be seen in Fig 3–1. The process starts when user captures a photo, that photo is then uploaded to a server which contains the deep learning models (OCR, scene description and chat box). Once the photo is received by the server, it is inferred through the neural networks. Each model generates its output and these outputs can be combined (text regions with detected objects) if there is a relevant match. A stepwise approach (see Fig 2–1) is followed in order to construct OCR pipeline to be invoked upon query. The scene text detection-recognition module consists of two major modules as is evident from its name.

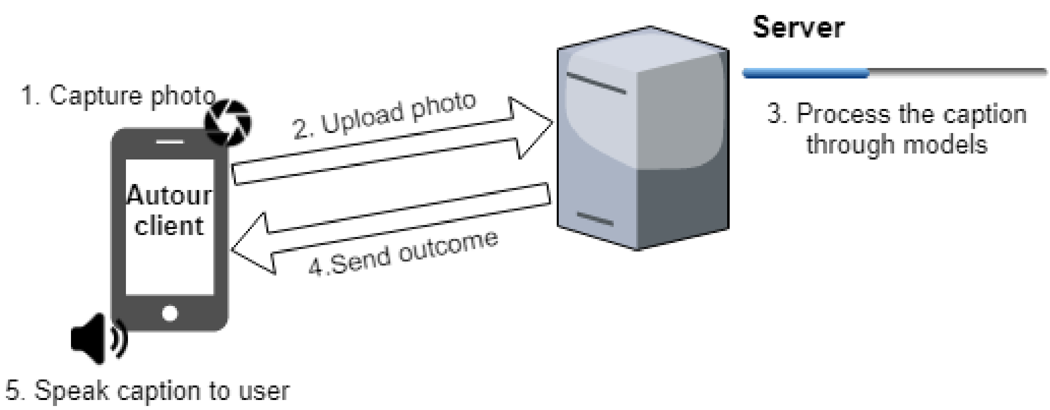

Figure 3–1: Autour's high-level data flow

3.1.1 Text detection

Text detection module relies on EAST: An Efficient and Accurate Scene Text Detector [69] and its reimplementation [1] with TensorFlow [1]. A high-level overview of text detection module is depicted in Figure 3–2. The fundamental element of the proposed algorithm is a fully-convolutional neural network which follows the general principles of [21]. The proposed model abandons unnecessary intermediate components and steps, and allows for end-to-end training and optimization. The resultant system, equipped with a single, light-weighted neural network, surpasses all previous methods by an obvious margin in both performance and speed.

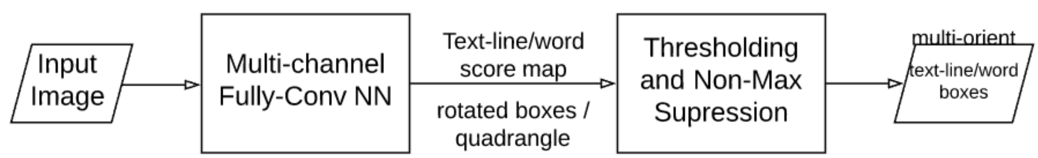

Figure 3–2: Text detection high-level data flow

Various parameters should be taken into consideration whilst designing neural networks aimed to detect text. Predicting bounding geometries of a small text region requires low level information in early stages, while large text regions need features from late-stage of a neural network. A network should be designed in a way that it should be able to utilize features from early and late stages of its structure, considering the fact that real scene text regions are not uniform. HyperNet [30] meets these conditions on features maps, however merging a large number of channels on large feature maps would significantly increase the computation overhead for later stages. In remedy of this, authors

[1] https://github.com/argman/EAST

of EAST have adopted the idea from U-shape [49] to merge feature maps gradually, while keeping the up-sampling branches small [69].

EAST's model can be decomposed in to three parts: feature extractor *stem*, feature-merging *branch* and output layer. An image is passed into the FCN, starting from the feature extractor and multiple channels of pixel-level text score map and geometry are generated at the output layer. A generic FCN architecture can be seen in Fig 3–3. The *stem* can be a convolutional network with interleaving convolution and pooling layers. Four levels of feature maps are extracted from the stem, whose sizes are respectively $\frac{1}{32}$, $\frac{1}{16}$, $\frac{1}{8}$ and $\frac{1}{4}$ of the input image. As a base model of stem, three different networks are examined: VGG16 [54] which is a commonly used model in many tasks, PVANET [19] a light-weight substitute of the feature extractor Faster-RCNN [48] and its double-channeled version *PVANET2x*. A generic FCN architecture developed for classification is depicted in Fig 3–3.

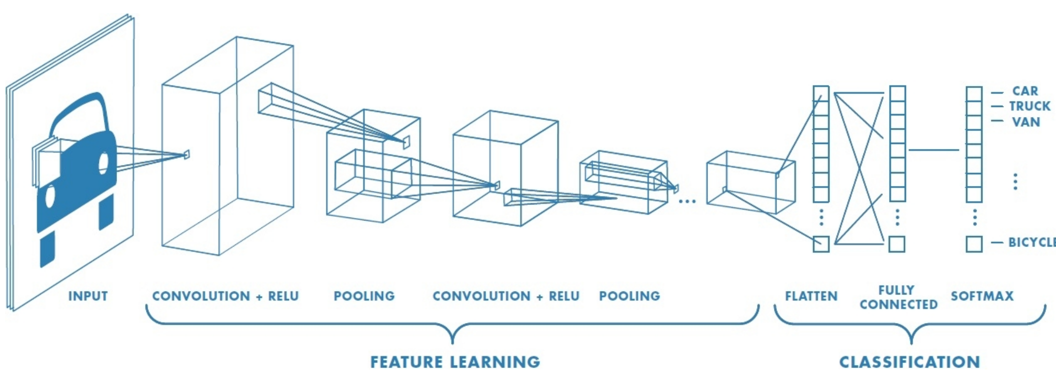

Figure 3–3: A generic FCN architecture
Image taken from from Matworks - https://www.mathworks.com/discovery/convolutional-neural-network.html

In the *branch* the features are gradually merged, in each merging stage, the feature map from the last stage is first fed to an unpooling layer to double its size, and then concatenated with the current feature map [69]. The final output layer contains several 1×1 convolution operations to project 32 channels of feature maps into 1 channel of score map and a multi-channel geometry map. Authors of EAST have experimented two geometry shapes for text regions, rotated box (RBOX) and quadrangle (QUAD), the geometry output can be one of them. For RBOX, the geometry is represented by 4 channels of axis-aligned bounding box (AABB) $\mathbf{R}$ and 1 channel rotation angle θ, $\mathbf{R}$ is formulated in [21]. The authors of [21] have defined the left top and right bottom points of the target bounding box in output coordinate space as $p_t = (x_t, y_t)$ and as $p_b = (x_b, y_b)$ respectively, then each pixel i is located at (x_i, y_i) in the output feature map describes a bounding box with a 5-dimensional vector as $\hat{t_i} = i\{\hat{s}, \hat{dx^t} = x_i - x_t, \hat{dy^t} = y_i - y_t, \hat{dx^b} = x_i - x_t, \hat{dy^b} = y_i - y_t\}$, 4 channels $\hat{dx^t}, \hat{dy^t}, \hat{dx^b}, \hat{dy^b}$ denote 4 distances from the pixel location to the top, right, bottom, left boundaries of the rectangle respectively and $\hat{s}$ is the confidence score of being an object in. In EAST, $\hat{s}$ denotes the rotation angle θ. For QUAD, 8 numbers are used to denote the coordinate shift from four corner vertices (same as RBOX description) of the quadrangle to the pixel location As each distance off-set contains two numbers $(\Delta x_i, \Delta y_i)$ the geometry output contains 8 channels [69].

Non-Maximum Suppression (NMS) stage has made its place in object detection methods as a post-processing step. As a popular pattern (object, text) detection approach the geometry scores are filtered by a predefined thresholding, valid geometries are then merged through NMS. A greedy NMS algorithm greedily selects high scoring detections and deletes close-by less confident neighbors since they are likely to cover the same object, a reasonable practice, since the actual goal is to generate exactly one detection per object. Yet the greedy NMS method makes hard decision by deleting detections and bases this decision on one fixed parameter that controls how wide the suppression is [20].

A naive NMS (running in $O(n^2)$, where number of candidate geometries is noted as n) would not be efficient, as the predictions are in great number. As an alternative a weighted merge algorithm is developed by the authors of EAST instead of a greedy NMS by assuming that closely located pixels are likely to be highly correlated. Developed algorithm merges the geometries row by row, and while merging geometries in the same row, it iteratively merges the geometry currently encountered with the last merged one. The coordinates of merged quadrangle are weight-averaged by the scores of two given quadrangle. To be clear, given two quadrangles g, p, the proposed method *WEIGHTEDMERGE* is defined as follows: $a = WEIGHTEDMERGE(g, p)$ then $a_i = V(g)g_i + V(p)_i$ and $V(a) = V(g) + V(p)$ where a_i is one of the coordinates of a subscripted by i and $V(a)$ is the score of geometry a.

3.1.2 Text recognition

Text recognition module is built on An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition [51] which originally is implemented in Torch [10] but for this project its PyTorch [47] [2] port [3] is chosen. Convolutional Recurrent Neural Network is explicitly designed for recognizing sequence-like objects in images. CRNN is a combination of Deep Convolutional Neural Network (DCNN) and Recurrent Neural Network (RNN). This combination possesses desired properties from both architecture and it is not confined by their constraints. Like a DCNN, CRNN is able to learn informative representations directly from image data but it has much less parameters than a standard DCNN model. CRNN is capable of producing sequence of labels still it is not constrained by the length

[2] http://pytorch.org/

[3] https://github.com/meijieru/crnn.pytorch

of the sequence-like object. A high-level overview of text detection module is depicted in Fig 3–4.

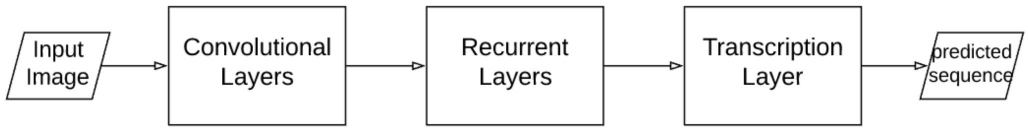

Figure 3–4: Text recognition high-level data flow

The convolutional layers automatically extract a feature sequence from each input image. These layers are adopted from a standard CNN by taking its convolutional and max-pooling layers and removing fully-connected layers so that it can be used to extract a sequential feature representation from an input image. Then a sequence of feature vectors is extracted from the feature maps produced by the convolutional layers and fed into recurrent layers. Convolutional layer architecture is based on well-known VGG [54] design. A deep bidirectional Recurrent Neural Network, as recurrent layers, is built on the top of the convolutional layers, in order to make prediction for each frame of the feature sequence. An RNN is favored in this context because of several reasons: its strong capability of capturing contextual information within a sequence, its ability to back-propagate error differentials to its input, its strength to operate on sequences of arbitrary lengths, traversing from starts to ends. However, traditional RNN units deteriorates from vanishing gradient problem which prevents the gradients *i.e.* weights of the network, from changing its value. Vanishing gradient problem confines the range of context the unit can store and causes difficulty to train the network. The basic structure of a recurrent neuron is shown in the left hand side of Fig 3–5, the unfolding of three time steps of the RNN is depicted in the right-hand side.

Long Short Term Memory (LSTM) [18] is specifically designed to address

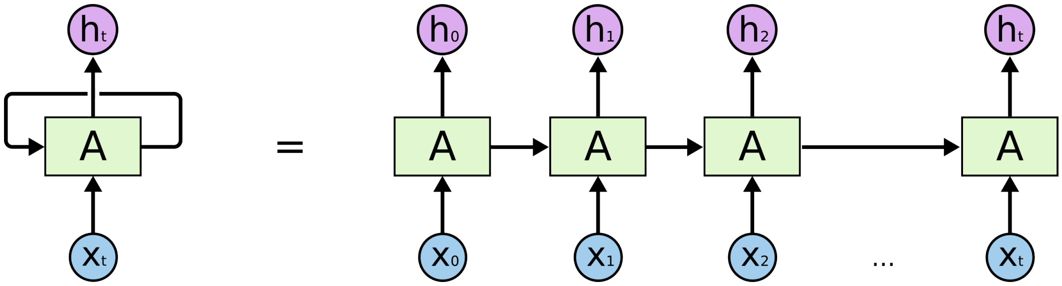

Figure 3–5: An RNN architecture
Image taken from from https://wiki.tum.de/display/lfdv/Recurrent+Neural+Networks+-+Combination+of+RNN+and+CNN

this problem. A common LSTM composition consists of a memory cell, an input gate, an output gate and a forget gate. Essentially, the memory cell stores past values *e.g.* states, and the input and output gates allow the cell to store values for either long or short time periods. This time interval is defined by designing the forget gate's activation, namely, the use of linear identity function (which has 1 as its derivative) will avoid the gradient to vanish. Nevertheless, in image-based sequences, information from both forward and backward directions are valuable and complementary to each other yet traditional LSTM's are unidirectional, allowing the use of only past information. To overcome this issue a bidirectional LSTM is designed by merging a forward and a backward LSTM. A generic LSTM is depicted in Fig 3–6 with its gates. The implementation can be seen on **crnn.py** on page 91 as class named *BidirectionalLSTM*.

The transcription layer, as the final component of CRNN is adopted to translate the per-frame predictions by the recurrent layers into a label sequence. For this reason conditional probability defined by Connectionist Temporal Classification (CTC) layer proposed in [14] is adopted . CTC is proposed to solve a fundamental issue of RNNs. RNNs can only be trained to make a series of independent label classifications. This means that the training data must be pre-segmented, and that the network outputs must be post-processed

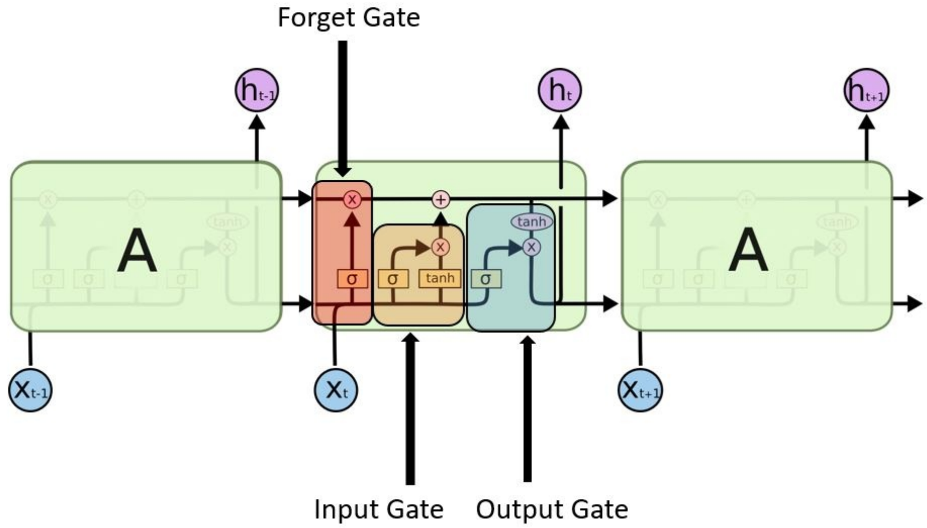

Figure 3–6: A generic LSTM architecture
Image taken from: https://wiki.tum.de/display/lfdv/Recurrent+Neural+Networks+-+Combination+of+RNN+and+CNN

to give the final label sequence. CTC models all aspects of the sequence within a single network architecture and trains the network to label the entire input sequence at once [14].

Both lexicon-based and lexicon-free transcription are examined. However, for extensive lexicons, such as Hunspell spell-checking dictionary[4] , it would be highly time consuming to execute an exhaustive search over the lexicon. The authors of [51] have discovered that label sequences predicted via lexicon-free transcription are generally close to the ground-truth under the edit distance metric. Exploiting this fact, CRNN authors were able to limit the search to the nearest neighbor candidates in order to cope with long search issue.

The objective function (negative log-likelihood of conditional probability of ground truth) which calculates a cost value directly from the image and its

[4] https://hunspell.github.io

ground truth label sequence is defined. Stochastic gradient algorithm is used to train the network, where gradients are calculated by the back-propagation algorithm [51].

3.2 Implementation

3.2.1 Text detection

The text detection module is implemented using Google's open-source machine learning framework TensorFlow. Main programming language of the project is Python. However, locality aware NMS module is written in C++ by the authors of [51] which provides drastic time savings. The project depends on several Python packages. SciPy, is a Python ecosystem consists of packages like NumPy and Matplotlib which are commonly used in mathematics, science, and engineering applications. Especially NumPy is a preferred Python library especially for scientific applications like neural networks, providing support for large, multi-dimensional arrays and matrices, along with a broad set of high-level mathematical functions to perform on these arrays. Polygon function of Shapely, a Python package for manipulation and analysis of planar geometric objects, ease area calculations of polygons, use can be seen on page 45 of **icdar.py**. Besides that there are several architectural differences between the implementation and the published paper: paper adopts three different base networks VGG16 [54], PVANET [19] and *PVANET2x* for its feature extractor *stem*, for the Tensorflow implementation of ResNet50 [16] is selected. ResNet (Residual Network) introduces residual learning, which eases the training of the network and provides increased depth. The variants of ResNet (ResNet50, ResNet101, ResNet152) have proved their success and accuracy on image classification. As ResNet V1 50 implementation [5] of TensorFlow's TF-Slim library,

[5] https://github.com/tensorflow/models/blob/master/research/slim/nets/

a lightweight package for defining, training and evaluating models, is utilized. The paper experiments with two geometry shapes for text regions: rotated box (RBOX) and quadrangle (QUAD). EAST authors have adopted balanced cross entropy loss inspired by [63] to facilitate a simpler training procedure. The loss function is defined as $-\beta Y^* log\hat{Y} - (1-\beta)(1-Y^*)log(1-\hat{Y})$ where $\hat{Y}$ is prediction of the score map (named *F_score* in the code), Y^* is the ground truth and β is the balancing factor between positive and negative samples. Alternatively the implementation uses dice loss (optimize Intersection over Union of segmentation) for both score map prediction and RBOX regression. Briefly, dice loss is a measure of overlap, proposed in [41] as a loss function. Authors of [41] states that dice coefficient is a quantity ranging between 0 and 1 aimed to be maximized. The dice coefficient D between two binary volumes is be written as:

$$D = \frac{2\sum_{i}^{N} p_i g_i}{\sum_{i}^{N} p_i^2 + \sum_{i}^{N} g_i^2}$$

where the sums run over the N voxels, of the predicted binary segmentation volume $p_i \in P$ and the ground truth binary volume $g_i \in G$. The code definition of loss function and dice coefficients can be seen on page 64, defined in **model.py** as *loss* and *dice_coefficient* functions respectively.

3.2.2 Text recognition

The text recognition module is originally implemented [6] by the authors of [51], using Torch [10] an open source machine learning library implemented in C with a wrapper in LuaJIT scripting language. The original implementation is a good reflection of the published paper with custom implementations

[6] https://github.com/bgshih/crnn

for the LSTM units (in Torch7/CUDA), the transcription layer (in C++) and the BK-tree data structure (in C++). However for Autour's OCR PyTorch port is selected, which is slightly different than the original. The original implementation contains two modes of transcription, namely the lexicon-free and lexicon-based transcriptions. In lexicon-free mode, predictions are made without any lexicon. In lexicon-based mode, predictions are made by choosing the label sequence that has the highest probability. In the paper, ADADELTA [67] is used for optimization to automatically calculate per-dimension learning rates. The PyTorch port is built on PyTorch, an open-source deep learning framework mainly developed by Facebook's artificial intelligence research team. It wraps the core Torch binaries in Python. The project has two dependencies: lmdb and warp-ctc. Lmdb is a universal Python binding for the LMDB Lightning Database, it is used to create a tiny database to manipulate (resize, label, batch) the images. Warp-CTC is a CTC implementation developed by Baidu's AI Lab. Since the project is implemented in PyTorch, Warp-CTC's PyTorch binding[7] is used. Unlike the original implementation PyTorch port does not provide lexicon-based transcriptions despite this it enables to choose between ADAM, ADADELTA and RMSprop as optimizer.

3.2.3 Pipelining

Testings of this project is started in McGill University's Shared Reality Lab computer with a dedicated NVIDIA Tesla K40c GPU with CUDA 7.5 installed and a 4 core Intel Core i5 Haswell CPU at 3.50GHz. Tesla series GPUs are targeted for high-performance computing applications such as stream processing, deep learning(both training and inference). Tesla K40c model has 2880 processor cores, 12GB of memory with type DDR5 SDRAM.

[7] https://github.com/SeanNaren/warp-ctc

GPUs differently from CPUs are designed to compute the same instructions in parallel, they might have thousands of cores, more computational units and a higher bandwidth to retrieve from memory. On the other hand, DNN's are engineered in a manner such that at each layer of the network thousands of identical artificial neurons perform the same computation (especially huge amounts of matrix multiplications). As a consequence, the structure of a DNN fits greatly with the sorts of computation that a GPU can efficiently (by parallelizing) perform. Exclusively NVIDIA is pioneering artificial intelligence with their specially designed hardware and their SDKs such as cuDNN, cuSPARSE and cuBLAS to support main deep learning frameworks *e.g.* TensorFlow, Keras and PyTorch. CUDA Toolkit of NVIDIA supplies a development environment for building high performance GPU-accelerated applications. With CUDA, developers are able to greatly fasten computing applications by exploiting the power of GPUs. In GPU-accelerated applications, CPU handles the sequential part of the workload since it is is optimized for single-threaded performance, while the computationally demanding fragment of the application runs on GPU cores in parallel. As the modules are implemented in different frameworks, it is critical to make sure that both frameworks will be able to access GPU for computations by installing their GPU-enabled versions.

The main contribution of this thesis is to combine the two aforementioned modules in a pipeline. By default, EAST's implementation is generating axis-aligned minimum bounding boxes (AABB) are wrapped around the text region. Namely, the text detection module produces bounding boxes in a following manner (x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4). Meanwhile CRNN's implementation uses Python Imaging Library (PIL) which accepts rectangles as input, in this case it was not possible to use text detection's output without modifying.

The solution is to produce bounding boxes using its (min(x), max(x), min(y), max(y)) coordinates which encloses the AABB generated by text detection. These bounding boxes are very likely to have bigger size as they have excess pixels with no text content, modified to fit the conditions. In other words, a box with coordinates (377, 117, 463, 120, 465, 130, 378, 150) is enlarged to (377, 465, 117, 150). Finally these boxes are cropped from the image, represented as NumPy ndarrays and passed to text recognizer (**recognize.py**), starting from line 87 of **eval.py** on page 74. However, text detection module intrinsically produces tightly wrapped bounding boxes around the text region, this causes separation of words which are meant to be read and/or pronounced together.

A straightforward search algorithm is developed to predict such text regions (to be combined), so that each cropped box can be passed in recognition module individually. This algorithm is called *match_check* (defined in **box_modifier.py** on page 74) and it basically looks for an overlap of bounding boxes over all possible permutations of two boxes. Since the boxes are enlarged, the probability of overlap of relevant boxes is naturally increased. This approach lowers the probability of false transcriptions by breaking word sequences. An image is shown in Fig 3–7 to provide better understanding of **box_modifier.py**. Green boxes are created by the text detection module, yet red boxes are cropped to pass into text recognizer then as the overlapping boxes (the ones at top right-hand corner and the ones at mid right-hand side) are combined together (after recognition) to form a bigger box.

Recognition module then transcripts each cropped box into a string. In order to have accurate and relevant results a spellchecking library PyEnchant is placed after recognition. Each string is run through the dictionary to check

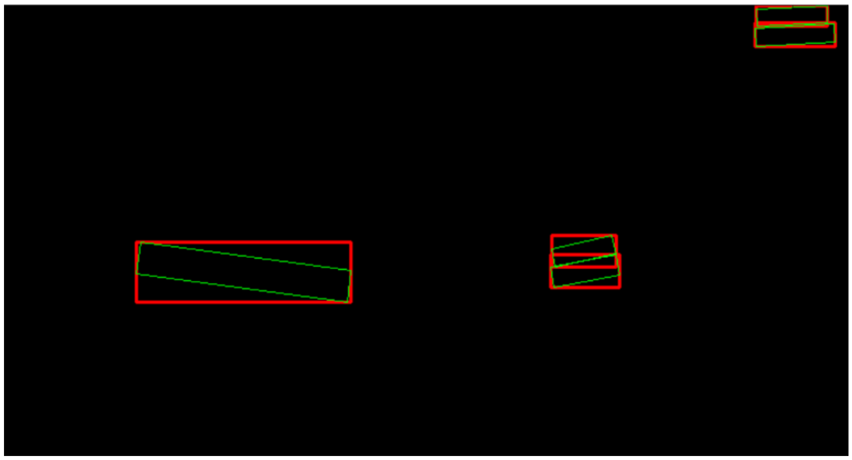

Figure 3–7: An example of box manipulation

whether the word is correctly spelled, in case the word is misspelt the first element from a suggestion list (where the suggested words are ordered from most likely replacement to least likely) is returned. It is an essential step, especially for a city like Montreal where both English and French is commonly used in daily life.

To sum up, this thesis work gathers two independent open-source work and builds a bridge between them. Output of the text detection module is manipulated and provided to text recognition, then the results are improved with the help of a spellchecking dictionary. Exploiting the location information of the boxes, the strings 'likely to be related' are combined with a developed algorithm and their box coordinates are merged. As the developed and provided projects of the authors of EAST [69] and CRNN [51] are performing satisfactory, no modifications are done on top of values selected by them. Python scripts of the original projects are divided into smaller parts in order to have a clear view and understanding e.g **preprocess.py**, **detect.py** and **recognize.py**. Project layout in an Integrated Development Environment can be seen on page 44. By default trained CRNN model is stored in directory *model_CRNN*, trained text detection (EAST) model is stored in directory *east_icdar2015_resnet_v1_50_rbox* and the input images are taken from directory *images*.

3.2.4 Inference

The inference can be initiated by invoking **eval.py** found on page 79. Multiple flags are set with *tf.app.flags.DEFINE_string* to define variables such as input data path, model path and GPU list. By its design Tensorflow has two main steps: constructing a dataflow graph (*tf.Graph*) and executing the graph in a session (*tf.Session*). Generally most Tensorflow programs begins with a dataflow construction stage, where nodes (*tf.Operation*) and edges (*tf.Tensor*) are specified in an abstract way. Then a *tf.Session* is created, running the session allows the graph to be executed on resources like CPU or GPU. This approach brings few favorable attributes such as parallelism, distributed execution, compilation and portability. In **eval.py** they are defined as *tf.get_default_graph().as_default()* and *tf.Session(config=tf.ConfigProto(allow_soft_placement=True)) as sess* respectively, *allow_soft_placement* flag ignores *tf.device* annotations that attempt to place CPU-only operations on a GPU device. When the session is created the trained model is restored from *checkpoint_path* defined with a flag.

The function *get_images* (defined at **preproc.py**, page 77), is used to load the images from *test_data_path*. Then the images are resized by the function *resize_image* (there is no certain input size, however by design it has to be multiple of 32 and less than 2400 pixels per vertex in order to confine GPU memory usage).

In line 65 of **eval.py** the session is run, the input images are fed into model. Within the function *model* (defined in **model.py**) the model is easily built through the use of argument scoping, *arg_scope*, provided by *tf.slim*. The function *model* takes images as input and generates score (*F_score*) for each geometry (*F_geometry*), where *F_geometry* is represented by 4 channel of axis aligned bounding box and 1 channel rotation angle.

The obtained scores and geometries are passed to function *detect* (defined at **detect.py**, on page) as parameters: score_map and geo_map. Besides these, *detect* has three different constant threshold parameters: score_map_thresh, box_thresh=0.2, nms_thres=0.6. They are thresholds for respectively: score map, box, non-maximum suppression. Score maps are filtered according to threshold value and detected boxes are sorted via the y-axis afterwards. The sorted boxes are then restored with function *restore_rectangle* (defined in **icdar.py**, on page 45) before NMS.

NMS stage (implemented in C++ and ported to project) is initiated at line 37 of **detect.py** by invoking the function *merge_quadrangle_n9*, alternatively there is an unused Python version of NMS is included within the project under the name **locality_aware_nms.py**. The script **adaptor.cpp** (can be seen on page 87) describes the bindings. The obtained bounding boxes with low scores are filtered by the average score map, differently from paper.

Acquired bounding box coordinates are checked for their coherence, starting at line 83 of **eval.py**. In the sequel, these boxes are cropped as mentioned in section 3.2.3 and fed into recognizer.

Each step before this point was a component of text detection, by calling **recognize.py** system switches to text recognition. When the function *recognize* is invoked an instance of CRNN model (defined in **crnn.py**, on page 91) is created depending on CUDA availability. Then the trained model is loaded from *model_path*. The defined alphabet (at line 13 of **recognize.py**) is then converted from string to labels by function *deode* in class *strLabelconverter* defined in **utils.py** (can be found on page 93). The cropped bounding box image is loaded, resized (to 100×32) and passed into a PyTorch variable with the same name (*image*). Raw predictions are attained after an inference through the network (see line 45 of **recognize.py**).

The inference of an high definition image through the text detection network takes less than 1.5 seconds and less than 50 millisecond at NMS stage (around 400 milliseconds when NMS Python is used), an inference through the whole pipeline takes less than 3 seconds on the mentioned computer of Shared Reality Lab. On a mid-end computer with no CUDA support and an Intel i5 CPU at 2.50GHz (model runs on CPU), text detection inference takes about 4 seconds and the whole pipeline inference runs around 6 seconds. Bearing in mind that, as GPU limitations are not explicitly set, deep learning frameworks allocates all GPU memory for the process whilst they run. The size of the input image and the number of text regions in the image plays a crucial role on inference timing as well as GPU performance .

3.2.5 Training

Since there are two different modules implemented on two different frameworks, they have to be trained separately. Text detection model can be trained by running **multigpu_train.py**. Several flags are set in the beginning of the script e.g. *batch_size_per_gpu*, *learning_rate*, *save_checkpoint_steps* and *save_summary_steps*. At the function *tower_loss* (starting at line 28 of *multigpu_train.py*) the inference graph is built by calling the model defined at **model.py**. *Model_loss* is computed with *loss* function defined in **model.py** and by adding regularization losses to *model_loss*, *total_loss* is computed (see line 35 of **multigpu_train.py**) In the *main* function, placeholders for input images and geometry maps are created global step, learning rate and optimizer is defined. The session *tf.Session* starts at line 137, and runs until loss is diverged, if not until the defined step number is reached.

To train the text detection model benchmark datasets (can be found in section 5.1) can be used. For training, a separate text file should be provided for each image in dataset. For instance, if the scene image is represented as img_1.jpg the corresponding text file should be named as img_1.txt. Each text region should be noted with its coordinates in format (x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4) followed by its text representation. The annotations are loaded by using *load_annotation* function described in **icdar.py**. A sample training image is depicted in Fig 3–8, and corresponding text file should be as follows:

```
377,117,463,117,465,130,378,130,Genaxis Theatre
493,115,519,115,519,131,493,131,[06]
374,155,409,155,409,170,374,170,###
492,151,551,151,551,170,492,170,62-03
376,198,422,198,422,212,376,212,Carpark
494,190,539,189,539,205,494,206,###
374,1,494,0,492,85,372,86,###
```

Figure 3–8: A sample image training image from ICDAR15

The network is trained end-to-end using only training images from ICDAR 2015 and ICDAR 2013 with ADAM [29] optimizer. Learning rate of ADAM starts from 1e-3, decays to one-tenth every 27300 mini batches, and stops at 1e-5 (staged learning rate decay), implementation instead accepts linear learning rate decay [69], can be seen on page 68. Such trained model achieves 77.32% recall and 84.66% precision rate on ICDAR 2015 Incidental Scene Text Detection Challenge It would be beneficial to keep in mind that the script **multigpu_train.py** is developed considering ICDAR images as input (**icdar.py** is developed specifically to adapt to **multigpu_train.py**).

Text recognition module can be trained by running **crnn_main.py** (can be seen on page 81). The script starts with flags e.g. *ngpu* (number of GPUs to use), *batchSize*, *niter* (number of epochs to train for) and optimizer option (ADAM, ADADELTA or RMSprop). The function *trainBatch* takes the CNN model, optimizer (set up starts at line 114 of **crnn_main.py**) and CTCLoss from warp-ctc as criterion, trains a batch and returns *cost* (defined at line 187) until the *niter* is reached. To train the text recognition model, benchmark datasets (can be found in section 5.2) can be used. An LMDB Lightning Database should be created by using function **dataset.py**, any kind of preprocessing e.g. transformations, tensorization and batching is handled by class *lmdbDataset* defined in the same script. The training of CRNN model takes little more than 2 days using MJSynth synthetic dataset introduced in [24].

CHAPTER 4
Results and Discussion

4.1 Results

This section aims to demonstrate how the pipeline works by showing its results. A photo of an hotel taken in downtown Montreal is used. The implementation of EAST authors generates cyan colored bounding boxes, yellow colored bounding boxes are cropped from the image (see Fig 4–1).

Figure 4–1: An instance of text detection

Figure 4–2: Cropped boxes from Fig 4–1

The raw output of the text recognition module is as following:

```
1--------2----1----6------ => 1216
s-------c--o--m-f-o-r-t--- => scomfort
c------o--m--f--o--r--t--- => comfort
s-------u---i--t--e---s--- => suites
s-----u---i--t--e---s----- => suites
h-------o-----t---e---l--- => hotel
e------------t--o----h---- => etoh
p-----e---r--i-o---d--ee-- => periode
i--n--t-e--r---d-iitt-e--- => interdite
2----------------z-------- => 2z
s--------o---u----n---d--- => sound
```

Note that '*scomfort*' is the output of bigger box containing 'comfort' string found at top left-hand corner of the image, '*etoh*' is the output of small box containing reflected 'hotel' string and '2z' is the output of graffiti written on pole. After spellchecking filtering, accent removal of French words and box merging the outcome is *['comfort suites', 'hotel', 'suites comfort', 'periode interdite', 'comfort suites', '1216']*, followed by the box centre coordinate in percentage with regards to the image. To be more clear, if the box centre is at (120, 200) where the image size is 480×600 the returned location will be (0.25, 0.2).

The project is maintained in this [1] repository, the trained ResNet model can downloaded from this [2] link, a complete trained EAST model can be found in this [3] Google drive folder and trained CRNN model can be found this [4] Dropbox folder.

4.2 Discussion

Currently, this OCR engine has its role within an existing mobile application. Depending on external factors, the application is able to give detailed visual description of users' scene with a tolerable latency in real-life scenarios. However, it is necessary to be aware of weaknesses of the system. It is clear that today's mobile platforms are not able to run such sized models (scene description and OCR in Autour's case) with their resource limitations. As an initial phase, Autour server runs these models sequentially. Namely, the scene description model is run then it is followed by OCR model. Even though it is a minor setback (which can be solved by assigning multiple resources or switching to cloud computing platforms enabling dynamic scaling), it introduces noticeable latency.

The current string combinations are made only in pairs. Namely, the system will fail to combine if there are three consecutive words present in the scene. Besides that, the filtering method, as it stands, relies on a spellchecking dictionary and the system has no apriori information about the language of

---

[1] https://github.com/heildever/AutourOCR

[2] download.tensorflow.org/models/resnet_v1_50_2016_08_28.tar.gz

[3] https://drive.google.com/file/d/0B3APw5BZJ67ETHNPaU9xUkVoV0U/

[4] https://www.dropbox.com/s/dboqjk20qjkpta3/crnn.pth?dl=0

the text (options may vary in multi-cultural cities) which means if the word is misspelt the system suggests French words only. One of the intended improvements is to make use of location information coming from Foursquare™, Google Places™ or OpenStreetMap look for possible matches i.e. in case of Fig 4–1 the hotel name 'Comfort Suites' can be confirmed by location information. As blind users of Autour indicated that they are not interested in every text in their scene.

The text detection module holds a prestigious ranking among published methods, and its implementation is performing satisfactory for Autour's use. However, ICDAR 2015 dataset is constructed by images containing mostly horizontally oriented text regions. Since the model is trained on such dataset, this may cause miss or imprecise predictions in case system encounters a vertically text instance.

CRNN inherently is able to recognize text sequences written with a straight orientation with similar characters. In the other words, system will fail to recognize if the text is written upside down or slightly angled. On the other hand, authors of CRNN [51] made a tweak to recognize English texts by adopting 1×2 sized rectangular pooling windows instead of the conventional squared one in the 3rd and 4th max-pooling layers. Assuming that typically a feature sequence of 25 frames can be generated from an image sized 100×32 and an image with that size contains up to 10 characters which exceeds the length of most English words. Meaning that system might misrecognize the words longer than 10 characters.

Nevertheless CRNN [51] is still accepted as state-of-the-art and used as a recognition module such as TextBoxes [35] and TextBoxes++ [34]. In addition to this the authors of CRNN have published a model which is able to recognize several types of irregular text, including perspective text and curved text [52].

Differently, a unified network is proposed in [37] for simultaneous detection and recognition. A new differentiable operator is introduced in order to share convolutional features between detection and recognition. Leveraging from convolution sharing strategy and the joint training method (enabling to learn more generic features), the method outperforms almost all state-of-the-art methods in both text localization and end-to-end tasks of ICDAR Incidental Scene Text challenge. Rankings can be seen on ICDAR's website [5] .

[5] http://rrc.cvc.uab.es/?ch=4&com=evaluation&task=1

CHAPTER 5
Benchmark Datasets

5.1 Text detection

COCO-Text [57] is a large scale dataset for text detection and recognition in natural images. The dataset is based on the MS COCO dataset, which contains images of complex everyday scenes. The dataset is organized around three tasks: Text localization, Cropped Word Recognition and End-to-End Recognition. The images were not collected with text in mind and thus contain a broad variety of text instances. The dataset contains 63,686 images with 173,589 labeled text regions in which 43,686 are chosen to be the training set and the rest 20,000 for testing. Three images from COCO-Text are shown in Fig 5–1.

Figure 5–1: Typical images from COCO-text

MSRA-TD500 [62] The MSRA Text Detection 500 Database contains 500 natural images, which are taken from indoor (office and mall) and outdoor (street) scenes using a pocket camera. The indoor images are mainly signs, doorplates and caution plates while the outdoor images are mostly guide boards and billboards in complex background. The dataset is divided into two parts: training set and test set. The training set contains 300 images randomly selected from the original dataset and the remaining 200 images constitute the test set. All the images in this dataset are fully annotated. Several images from MSRA-TD500 are depicted in Fig 5–2, text regions are emphasized with bounding boxes.

Figure 5–2: Typical images from MSRA-TD500

ICDAR 2015 [27] Competition on Robust Reading is structured in four challenges addressing text extraction in different application domains, namely born-digital images, focused scene images, incidental scene text and video text. Incidental Scene Text refers to text that appears in the scene without the user having taken any prior action to cause its appearance in the field of view, or improve its positioning or quality in the frame. While focused scene text is the expected input for applications such as translation on demand, incidental scene text covers another wide range of applications linked to wearable cameras or massive urban captures where the capture is difficult or undesirable to control. For incidental scene text a new dataset is introduced, a dataset of 1,670 images (17,548 annotated regions) acquired using the Google Glass.

Authors of [27] have used 1000 of images are used for training and the remaining are for testing. Few images from ICDAR'S incidental scene text dataset are shown in Fig 5–3.

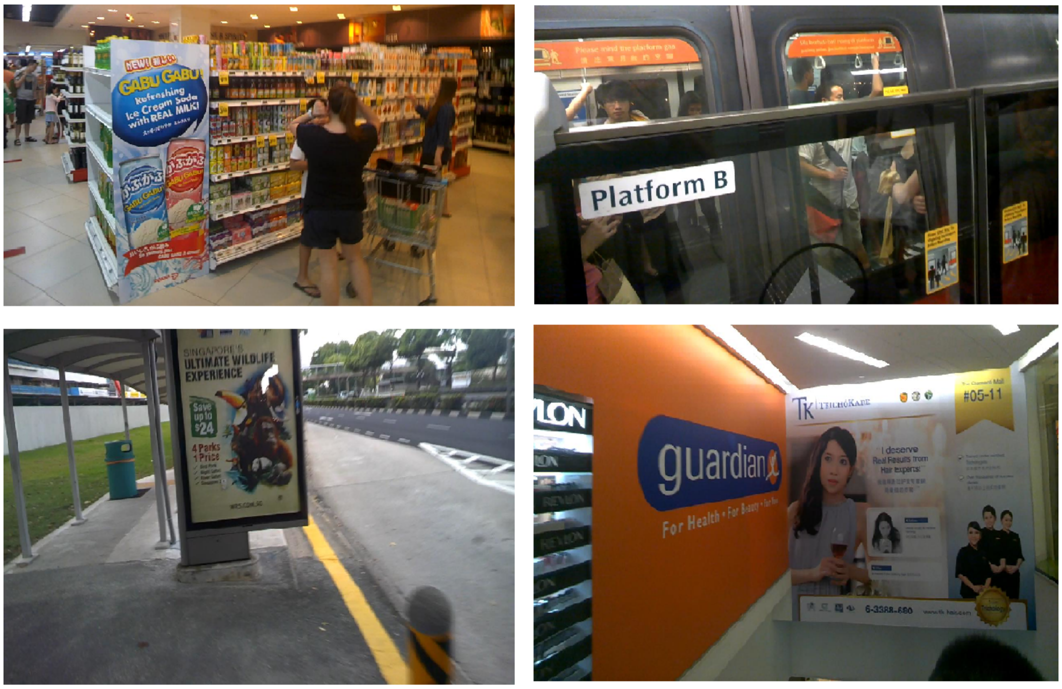

Figure 5–3: Typical images from ICDAR incidental scene text dataset

5.2 Text recognition

IIIT 5K [42] is harvested from Google image search. Query words like billboards, signboard, house numbers, house name plates, movie posters were used to collect images. The dataset contains 5000 cropped word images from Scene Texts and born-digital images. The dataset is divided into train and test parts. Three training images (above) and three test images (below) of IIIT5K are depicted in Fig 5–4.

SVT [58] The Street View Text dataset was harvested from Google Street View. The annotators were asked to find a place of business with its signboard, then have a clear point of view to minimize the skew of the text before taking a screen shot. Therefore the dataset includes business names and business

Figure 5–4: Typical train images IIIT

signs. Each business (hotel, restaurant and etc) and its sign is associated with representative text. Precisely; the annotators have generated annotations of horizontal bounding boxes and a non case-sensitive transcription for each text region. The word annotations are adopted to produce a dataset of cropped words called SVT-50. In total, SVT dataset contains 100 training and 250 testing images collected from 20 different cities. Four different SVT images can be seen in Fig 5–5.

IC03 datasets are created by the images that have been extracted from natural scenes for the ICDAR 2003 Robust Reading competitions. Four independent competitions were organized: Robust Reading, Robust Word Recognition, Robust Character Recognition and Text Locating. Recognition datasets are word recognition (made of 1157 words training set, 1111 words of testing set) and character recognition (made of 6185 characters training set, 5430 characters testing set). Few cropped character and cropped word images are depicted in Fig 5–6.

IC11 and IC13, IC11 is an extension of datasets used in earlier Robust Reading Competitions organized in ICDAR 2003 and 2005. Images of IC03 are taken as a base, the images without text are removed and around 100 images are added which are captured with a digital camera using auto focus and natural lighting. The final dataset consisted of 485 images containing

Figure 5–5: Typical images from SVT

text in a variety of colors and fonts on many different backgrounds and in various orientations. IC13 is an image dataset used for the ICDAR2013 Robust Reading Competition is almost the same as the dataset IC11. The difference from the ICDAR2011 dataset is revision of ground-truth texts at several images. In addition, a small number of images duplicated over training and test sets were excluded. Accordingly, ICDAR2013(IC13) dataset is a subset of ICDAR2011(IC11) dataset. The number of images of ICDAR2013 dataset is 462, which is comprised of 229 images for the training set and 233 images for the test set. Cropped characters (above) and cropped words (below) are shown in Fig 5–6.

Figure 5–6: Typical character recognition images IC03

Figure 5–7: Typical images from IC13

CHAPTER 6
Appendices

Appendix A - project layout

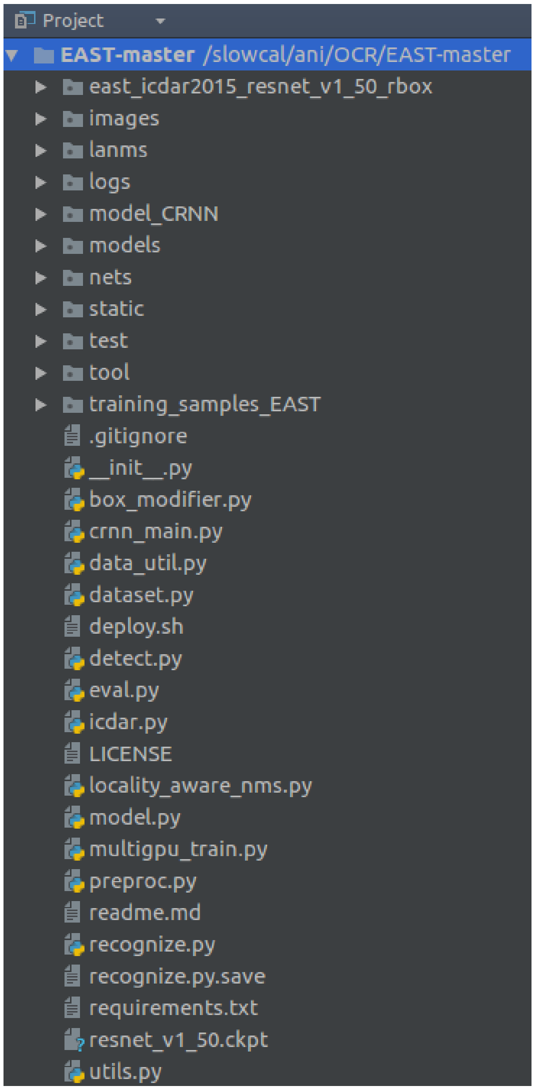

Figure 6–1: Project layout

Appendix B - icdar.py

```python
# coding:utf-8
import glob
import csv
import cv2
import time
import os
import numpy as np
import scipy.optimize
import matplotlib.pyplot as plt
import matplotlib.patches as Patches
from shapely.geometry import Polygon

import tensorflow as tf

from data_util import GeneratorEnqueuer

tf.app.flags.DEFINE_string('training_data_path', '/data/ocr/icdar2015/',
                           'training dataset to use')
tf.app.flags.DEFINE_integer('max_image_large_side', 1280,
                            'max image size of training')
tf.app.flags.DEFINE_integer('max_text_size', 800,
                            'if the text in the input image is bigger than this, then we resize'
                            'the image according to this')
tf.app.flags.DEFINE_integer('min_text_size', 10,
                            'if the text size is smaller than this, we ignore it during training')
tf.app.flags.DEFINE_float('min_crop_side_ratio', 0.1,
                          'when doing random crop from input image, the'
                          'min length of min(H, W')
tf.app.flags.DEFINE_string('geometry', 'RBOX',
                           'which geometry to generate, RBOX or QUAD')


FLAGS = tf.app.flags.FLAGS


def get_images():
```

```python
    files = []
    for ext in ['jpg', 'png', 'jpeg', 'JPG']:
        files.extend(glob.glob(
            os.path.join(FLAGS.training_data_path, '*.{}'.format(ext))))
    return files


def load_annotation(p):
    '''
    load annotation from the text file
    :param p:
    :return:
    '''
    text_polys = []
    text_tags = []
    if not os.path.exists(p):
        return np.array(text_polys, dtype=np.float32)
    with open(p, 'r') as f:
        reader = csv.reader(f)
        for line in reader:
            label = line[-1]
            # strip BOM. \ufeff for python3,  \xef\xbb\bf for python2
            line = [i.strip('\ufeff').strip('\xef\xbb\xbf') for i in line]

            x1, y1, x2, y2, x3, y3, x4, y4 = list(map(float, line[:8]))
            text_polys.append([[x1, y1], [x2, y2], [x3, y3], [x4, y4]])
            if label == '*' or label == '###':
                text_tags.append(True)
            else:
                text_tags.append(False)
        return np.array(text_polys, dtype=np.float32), np.array(text_tags, dtype=np.bool)


def polygon_area(poly):
    '''
    compute area of a polygon
    :param poly:
    :return:
    '''
    edge = [
        (poly[1][0] - poly[0][0]) * (poly[1][1] + poly[0][1]),
```

```
        (poly[2][0] - poly[1][0]) * (poly[2][1] + poly[1][1]),
        (poly[3][0] - poly[2][0]) * (poly[3][1] + poly[2][1]),
        (poly[0][0] - poly[3][0]) * (poly[0][1] + poly[3][1])
    ]
    return np.sum(edge)/2.


def check_and_validate_polys(polys, tags, xxx_todo_changeme):
    '''
    check so that the text poly is in the same direction,
    and also filter some invalid polygons
    :param polys:
    :param tags:
    :return:
    '''
    (h, w) = xxx_todo_changeme
    if polys.shape[0] == 0:
        return polys
    polys[:, :, 0] = np.clip(polys[:, :, 0], 0, w-1)
    polys[:, :, 1] = np.clip(polys[:, :, 1], 0, h-1)

    validated_polys = []
    validated_tags = []
    for poly, tag in zip(polys, tags):
        p_area = polygon_area(poly)
        if abs(p_area) < 1:
            # print poly
            print('invalid poly')
            continue
        if p_area > 0:
            print('poly in wrong direction')
            poly = poly[(0, 3, 2, 1), :]
        validated_polys.append(poly)
        validated_tags.append(tag)
    return np.array(validated_polys), np.array(validated_tags)


def crop_area(im, polys, tags, crop_background=False, max_tries=50):
    '''
    make random crop from the input image
    :param im:
```

```python
    :param polys:
    :param tags:
    :param crop_background:
    :param max_tries:
    :return:
    '''
    h, w, _ = im.shape
    pad_h = h//10
    pad_w = w//10
    h_array = np.zeros((h + pad_h*2), dtype=np.int32)
    w_array = np.zeros((w + pad_w*2), dtype=np.int32)
    for poly in polys:
        poly = np.round(poly, decimals=0).astype(np.int32)
        minx = np.min(poly[:, 0])
        maxx = np.max(poly[:, 0])
        w_array[minx+pad_w:maxx+pad_w] = 1
        miny = np.min(poly[:, 1])
        maxy = np.max(poly[:, 1])
        h_array[miny+pad_h:maxy+pad_h] = 1
    # ensure the cropped area not across a text
    h_axis = np.where(h_array == 0)[0]
    w_axis = np.where(w_array == 0)[0]
    if len(h_axis) == 0 or len(w_axis) == 0:
        return im, polys, tags
    for i in range(max_tries):
        xx = np.random.choice(w_axis, size=2)
        xmin = np.min(xx) - pad_w
        xmax = np.max(xx) - pad_w
        xmin = np.clip(xmin, 0, w-1)
        xmax = np.clip(xmax, 0, w-1)
        yy = np.random.choice(h_axis, size=2)
        ymin = np.min(yy) - pad_h
        ymax = np.max(yy) - pad_h
        ymin = np.clip(ymin, 0, h-1)
        ymax = np.clip(ymax, 0, h-1)
        if xmax - xmin < FLAGS.min_crop_side_ratio*w or ymax - ymin < FLAGS.min_crop_side_ratio*h:
            # area too small
            continue
        if polys.shape[0] != 0:
            poly_axis_in_area = (polys[:, :, 0] >= xmin) & (polys[:, :, 0] <= xmax) \
```

```python
                                & (polys[:, :, 1] >= ymin) & (polys[:, :, 1] <= ymax)
            selected_polys = np.where(np.sum(poly_axis_in_area, axis=1) == 4)[0]
        else:
            selected_polys = []
        if len(selected_polys) == 0:
            # no text in this area
            if crop_background:
                return im[ymin:ymax+1, xmin:xmax+1, :], polys[selected_polys], tags[selected_polys]
            else:
                continue
        im = im[ymin:ymax+1, xmin:xmax+1, :]
        polys = polys[selected_polys]
        tags = tags[selected_polys]
        polys[:, :, 0] -= xmin
        polys[:, :, 1] -= ymin
        return im, polys, tags

    return im, polys, tags


def shrink_poly(poly, r):
    '''
    fit a poly inside the origin poly, maybe bugs here...
    used for generate the score map
    :param poly: the text poly
    :param r: r in the paper
    :return: the shrinked poly
    '''
    # shrink ratio
    R = 0.3
    # find the longer pair
    if np.linalg.norm(poly[0] - poly[1]) + np.linalg.norm(poly[2] - poly[3]) > \
                    np.linalg.norm(poly[0] - poly[3]) + np.linalg.norm(poly[1] - poly[2]):
        # first move (p0, p1), (p2, p3), then (p0, p3), (p1, p2)
        ## p0, p1
        theta = np.arctan2((poly[1][1] - poly[0][1]), (poly[1][0] - poly[0][0]))
        poly[0][0] += R * r[0] * np.cos(theta)
        poly[0][1] += R * r[0] * np.sin(theta)
        poly[1][0] -= R * r[1] * np.cos(theta)
        poly[1][1] -= R * r[1] * np.sin(theta)
```

```
        ## p2, p3
        theta = np.arctan2((poly[2][1] - poly[3][1]), (poly[2][0] - poly[3][0]))
        poly[3][0] += R * r[3] * np.cos(theta)
        poly[3][1] += R * r[3] * np.sin(theta)
        poly[2][0] -= R * r[2] * np.cos(theta)
        poly[2][1] -= R * r[2] * np.sin(theta)
        ## p0, p3
        theta = np.arctan2((poly[3][0] - poly[0][0]), (poly[3][1] - poly[0][1]))
        poly[0][0] += R * r[0] * np.sin(theta)
        poly[0][1] += R * r[0] * np.cos(theta)
        poly[3][0] -= R * r[3] * np.sin(theta)
        poly[3][1] -= R * r[3] * np.cos(theta)
        ## p1, p2
        theta = np.arctan2((poly[2][0] - poly[1][0]), (poly[2][1] - poly[1][1]))
        poly[1][0] += R * r[1] * np.sin(theta)
        poly[1][1] += R * r[1] * np.cos(theta)
        poly[2][0] -= R * r[2] * np.sin(theta)
        poly[2][1] -= R * r[2] * np.cos(theta)
    else:
        ## p0, p3
        # print poly
        theta = np.arctan2((poly[3][0] - poly[0][0]), (poly[3][1] - poly[0][1]))
        poly[0][0] += R * r[0] * np.sin(theta)
        poly[0][1] += R * r[0] * np.cos(theta)
        poly[3][0] -= R * r[3] * np.sin(theta)
        poly[3][1] -= R * r[3] * np.cos(theta)
        ## p1, p2
        theta = np.arctan2((poly[2][0] - poly[1][0]), (poly[2][1] - poly[1][1]))
        poly[1][0] += R * r[1] * np.sin(theta)
        poly[1][1] += R * r[1] * np.cos(theta)
        poly[2][0] -= R * r[2] * np.sin(theta)
        poly[2][1] -= R * r[2] * np.cos(theta)
        ## p0, p1
        theta = np.arctan2((poly[1][1] - poly[0][1]), (poly[1][0] - poly[0][0]))
        poly[0][0] += R * r[0] * np.cos(theta)
        poly[0][1] += R * r[0] * np.sin(theta)
        poly[1][0] -= R * r[1] * np.cos(theta)
        poly[1][1] -= R * r[1] * np.sin(theta)
        ## p2, p3
        theta = np.arctan2((poly[2][1] - poly[3][1]), (poly[2][0] - poly[3][0]))
        poly[3][0] += R * r[3] * np.cos(theta)
```

```python
        poly[3][1] += R * r[3] * np.sin(theta)
        poly[2][0] -= R * r[2] * np.cos(theta)
        poly[2][1] -= R * r[2] * np.sin(theta)
    return poly


def point_dist_to_line(p1, p2, p3):
    # compute the distance from p3 to p1-p2
    return np.linalg.norm(np.cross(p2 - p1, p1 - p3)) / np.linalg.norm(p2 - p1)


def fit_line(p1, p2):
    # fit a line ax+by+c = 0
    if p1[0] == p1[1]:
        return [1., 0., -p1[0]]
    else:
        [k, b] = np.polyfit(p1, p2, deg=1)
        return [k, -1., b]


def line_cross_point(line1, line2):
    # line1 0= ax+by+c, compute the cross point of line1 and line2
    if line1[0] != 0 and line1[0] == line2[0]:
        print('Cross point does not exist')
        return None
    if line1[0] == 0 and line2[0] == 0:
        print('Cross point does not exist')
        return None
    if line1[1] == 0:
        x = -line1[2]
        y = line2[0] * x + line2[2]
    elif line2[1] == 0:
        x = -line2[2]
        y = line1[0] * x + line1[2]
    else:
        k1, _, b1 = line1
        k2, _, b2 = line2
        x = -(b1-b2)/(k1-k2)
        y = k1*x + b1
    return np.array([x, y], dtype=np.float32)

```

```python

def line_verticle(line, point):
    # get the verticle line from line across point
    if line[1] == 0:
        verticle = [0, -1, point[1]]
    else:
        if line[0] == 0:
            verticle = [1, 0, -point[0]]
        else:
            verticle = [-1./line[0], -1, point[1] - (-1/line[0] * point[0])]
    return verticle


def rectangle_from_parallelogram(poly):
    '''
    fit a rectangle from a parallelogram
    :param poly:
    :return:
    '''
    p0, p1, p2, p3 = poly
    angle_p0 = np.arccos(np.dot(p1-p0, p3-p0)/(np.linalg.norm(p0-p1) *
     ↪  np.linalg.norm(p3-p0)))
    if angle_p0 < 0.5 * np.pi:
        if np.linalg.norm(p0 - p1) > np.linalg.norm(p0-p3):
            # p0 and p2
            ## p0
            p2p3 = fit_line([p2[0], p3[0]], [p2[1], p3[1]])
            p2p3_verticle = line_verticle(p2p3, p0)

            new_p3 = line_cross_point(p2p3, p2p3_verticle)
            ## p2
            p0p1 = fit_line([p0[0], p1[0]], [p0[1], p1[1]])
            p0p1_verticle = line_verticle(p0p1, p2)

            new_p1 = line_cross_point(p0p1, p0p1_verticle)
            return np.array([p0, new_p1, p2, new_p3], dtype=np.float32)
        else:
            p1p2 = fit_line([p1[0], p2[0]], [p1[1], p2[1]])
            p1p2_verticle = line_verticle(p1p2, p0)

            new_p1 = line_cross_point(p1p2, p1p2_verticle)
```

```
            p0p3 = fit_line([p0[0], p3[0]], [p0[1], p3[1]])
            p0p3_verticle = line_verticle(p0p3, p2)

            new_p3 = line_cross_point(p0p3, p0p3_verticle)
            return np.array([p0, new_p1, p2, new_p3], dtype=np.float32)
    else:
        if np.linalg.norm(p0-p1) > np.linalg.norm(p0-p3):
            # p1 and p3
            ## p1
            p2p3 = fit_line([p2[0], p3[0]], [p2[1], p3[1]])
            p2p3_verticle = line_verticle(p2p3, p1)

            new_p2 = line_cross_point(p2p3, p2p3_verticle)
            ## p3
            p0p1 = fit_line([p0[0], p1[0]], [p0[1], p1[1]])
            p0p1_verticle = line_verticle(p0p1, p3)

            new_p0 = line_cross_point(p0p1, p0p1_verticle)
            return np.array([new_p0, p1, new_p2, p3], dtype=np.float32)
        else:
            p0p3 = fit_line([p0[0], p3[0]], [p0[1], p3[1]])
            p0p3_verticle = line_verticle(p0p3, p1)

            new_p0 = line_cross_point(p0p3, p0p3_verticle)
            p1p2 = fit_line([p1[0], p2[0]], [p1[1], p2[1]])
            p1p2_verticle = line_verticle(p1p2, p3)

            new_p2 = line_cross_point(p1p2, p1p2_verticle)
            return np.array([new_p0, p1, new_p2, p3], dtype=np.float32)


def sort_rectangle(poly):
    # sort the four coordinates of the polygon, points in poly should be sorted clockwise
    # First find the lowest point
    p_lowest = np.argmax(poly[:, 1])
    if np.count_nonzero(poly[:, 1] == poly[p_lowest, 1]) == 2:
        # X, p0
        p0_index = np.argmin(np.sum(poly, axis=1))
        p1_index = (p0_index + 1) % 4
        p2_index = (p0_index + 2) % 4
        p3_index = (p0_index + 3) % 4
```

```
        return poly[[p0_index, p1_index, p2_index, p3_index]], 0.
    else:
        #
        p_lowest_right = (p_lowest - 1) % 4
        p_lowest_left = (p_lowest + 1) % 4
        angle = np.arctan(-(poly[p_lowest][1] - poly[p_lowest_right][1])/(poly[p_lowest][0] - poly[p_lowest_right][0]))
        # assert angle > 0
        if angle <= 0:
            print(angle, poly[p_lowest], poly[p_lowest_right])
        if angle/np.pi * 180 > 45:
            # p2
            p2_index = p_lowest
            p1_index = (p2_index - 1) % 4
            p0_index = (p2_index - 2) % 4
            p3_index = (p2_index + 1) % 4
            return poly[[p0_index, p1_index, p2_index, p3_index]], -(np.pi/2 - angle)
        else:
            # p3
            p3_index = p_lowest
            p0_index = (p3_index + 1) % 4
            p1_index = (p3_index + 2) % 4
            p2_index = (p3_index + 3) % 4
            return poly[[p0_index, p1_index, p2_index, p3_index]], angle


def restore_rectangle_rbox(origin, geometry):
    d = geometry[:, :4]
    angle = geometry[:, 4]
    # for angle > 0
    origin_0 = origin[angle >= 0]
    d_0 = d[angle >= 0]
    angle_0 = angle[angle >= 0]
    if origin_0.shape[0] > 0:
        p = np.array([np.zeros(d_0.shape[0]), -d_0[:, 0] - d_0[:, 2],
                      d_0[:, 1] + d_0[:, 3], -d_0[:, 0] - d_0[:, 2],
                      d_0[:, 1] + d_0[:, 3], np.zeros(d_0.shape[0]),
                      np.zeros(d_0.shape[0]), np.zeros(d_0.shape[0]),
                      d_0[:, 3], -d_0[:, 2]])
        p = p.transpose((1, 0)).reshape((-1, 5, 2))  # N*5*2

```

```
            rotate_matrix_x = np.array([np.cos(angle_0), np.sin(angle_0)]).transpose((1, 0))
            rotate_matrix_x = np.repeat(rotate_matrix_x, 5, axis=1).reshape(-1, 2,
             ↪   5).transpose((0, 2, 1))  # N*5*2

            rotate_matrix_y = np.array([-np.sin(angle_0), np.cos(angle_0)]).transpose((1, 0))
            rotate_matrix_y = np.repeat(rotate_matrix_y, 5, axis=1).reshape(-1, 2,
             ↪   5).transpose((0, 2, 1))

            p_rotate_x = np.sum(rotate_matrix_x * p, axis=2)[:, :, np.newaxis]  # N*5*1
            p_rotate_y = np.sum(rotate_matrix_y * p, axis=2)[:, :, np.newaxis]  # N*5*1

            p_rotate = np.concatenate([p_rotate_x, p_rotate_y], axis=2)  # N*5*2

            p3_in_origin = origin_0 - p_rotate[:, 4, :]
            new_p0 = p_rotate[:, 0, :] + p3_in_origin  # N*2
            new_p1 = p_rotate[:, 1, :] + p3_in_origin
            new_p2 = p_rotate[:, 2, :] + p3_in_origin
            new_p3 = p_rotate[:, 3, :] + p3_in_origin

            new_p_0 = np.concatenate([new_p0[:, np.newaxis, :], new_p1[:, np.newaxis, :],
                                      new_p2[:, np.newaxis, :], new_p3[:, np.newaxis, :]],
                                       ↪   axis=1)  # N*4*2
        else:
            new_p_0 = np.zeros((0, 4, 2))
        # for angle < 0
        origin_1 = origin[angle < 0]
        d_1 = d[angle < 0]
        angle_1 = angle[angle < 0]
        if origin_1.shape[0] > 0:
            p = np.array([-d_1[:, 1] - d_1[:, 3], -d_1[:, 0] - d_1[:, 2],
                          np.zeros(d_1.shape[0]), -d_1[:, 0] - d_1[:, 2],
                          np.zeros(d_1.shape[0]), np.zeros(d_1.shape[0]),
                          -d_1[:, 1] - d_1[:, 3], np.zeros(d_1.shape[0]),
                          -d_1[:, 1], -d_1[:, 2]])
            p = p.transpose((1, 0)).reshape((-1, 5, 2))  # N*5*2

            rotate_matrix_x = np.array([np.cos(-angle_1), -np.sin(-angle_1)]).transpose((1, 0))
            rotate_matrix_x = np.repeat(rotate_matrix_x, 5, axis=1).reshape(-1, 2,
             ↪   5).transpose((0, 2, 1))  # N*5*2

            rotate_matrix_y = np.array([np.sin(-angle_1), np.cos(-angle_1)]).transpose((1, 0))
```

```
        rotate_matrix_y = np.repeat(rotate_matrix_y, 5, axis=1).reshape(-1, 2,
         ↪   5).transpose((0, 2, 1))

        p_rotate_x = np.sum(rotate_matrix_x * p, axis=2)[:, :, np.newaxis]  # N*5*1
        p_rotate_y = np.sum(rotate_matrix_y * p, axis=2)[:, :, np.newaxis]  # N*5*1

        p_rotate = np.concatenate([p_rotate_x, p_rotate_y], axis=2)  # N*5*2

        p3_in_origin = origin_1 - p_rotate[:, 4, :]
        new_p0 = p_rotate[:, 0, :] + p3_in_origin  # N*2
        new_p1 = p_rotate[:, 1, :] + p3_in_origin
        new_p2 = p_rotate[:, 2, :] + p3_in_origin
        new_p3 = p_rotate[:, 3, :] + p3_in_origin

        new_p_1 = np.concatenate([new_p0[:, np.newaxis, :], new_p1[:, np.newaxis, :],
                                  new_p2[:, np.newaxis, :], new_p3[:, np.newaxis, :]],
                                   ↪   axis=1)  # N*4*2
    else:
        new_p_1 = np.zeros((0, 4, 2))
    return np.concatenate([new_p_0, new_p_1])


def restore_rectangle(origin, geometry):
    return restore_rectangle_rbox(origin, geometry)


def generate_rbox(im_size, polys, tags):
    h, w = im_size
    poly_mask = np.zeros((h, w), dtype=np.uint8)
    score_map = np.zeros((h, w), dtype=np.uint8)
    geo_map = np.zeros((h, w, 5), dtype=np.float32)
    # mask used during traning, to ignore some hard areas
    training_mask = np.ones((h, w), dtype=np.uint8)
    for poly_idx, poly_tag in enumerate(zip(polys, tags)):
        poly = poly_tag[0]
        tag = poly_tag[1]

        r = [None, None, None, None]
        for i in range(4):
            r[i] = min(np.linalg.norm(poly[i] - poly[(i + 1) % 4]),
                       np.linalg.norm(poly[i] - poly[(i - 1) % 4]))
```

```
        # score map
        shrinked_poly = shrink_poly(poly.copy(), r).astype(np.int32)[np.newaxis, :, :]
        cv2.fillPoly(score_map, shrinked_poly, 1)
        cv2.fillPoly(poly_mask, shrinked_poly, poly_idx + 1)
        # if the poly is too small, then ignore it during training
        poly_h = min(np.linalg.norm(poly[0] - poly[3]), np.linalg.norm(poly[1] - poly[2]))
        poly_w = min(np.linalg.norm(poly[0] - poly[1]), np.linalg.norm(poly[2] - poly[3]))
        if min(poly_h, poly_w) < FLAGS.min_text_size:
            cv2.fillPoly(training_mask, poly.astype(np.int32)[np.newaxis, :, :], 0)
        if tag:
            cv2.fillPoly(training_mask, poly.astype(np.int32)[np.newaxis, :, :], 0)

        xy_in_poly = np.argwhere(poly_mask == (poly_idx + 1))
        # if geometry == 'RBOX':
        #
        fitted_parallelograms = []
        for i in range(4):
            p0 = poly[i]
            p1 = poly[(i + 1) % 4]
            p2 = poly[(i + 2) % 4]
            p3 = poly[(i + 3) % 4]
            edge = fit_line([p0[0], p1[0]], [p0[1], p1[1]])
            backward_edge = fit_line([p0[0], p3[0]], [p0[1], p3[1]])
            forward_edge = fit_line([p1[0], p2[0]], [p1[1], p2[1]])
            if point_dist_to_line(p0, p1, p2) > point_dist_to_line(p0, p1, p3):
                # p2
                if edge[1] == 0:
                    edge_opposite = [1, 0, -p2[0]]
                else:
                    edge_opposite = [edge[0], -1, p2[1] - edge[0] * p2[0]]
            else:
                # p3
                if edge[1] == 0:
                    edge_opposite = [1, 0, -p3[0]]
                else:
                    edge_opposite = [edge[0], -1, p3[1] - edge[0] * p3[0]]
            # move forward edge
            new_p0 = p0
            new_p1 = p1
            new_p2 = p2
            new_p3 = p3
```

```
            new_p2 = line_cross_point(forward_edge, edge_opposite)
            if point_dist_to_line(p1, new_p2, p0) > point_dist_to_line(p1, new_p2, p3):
                # across p0
                if forward_edge[1] == 0:
                    forward_opposite = [1, 0, -p0[0]]
                else:
                    forward_opposite = [forward_edge[0], -1, p0[1] - forward_edge[0] * p0[0]]
            else:
                # across p3
                if forward_edge[1] == 0:
                    forward_opposite = [1, 0, -p3[0]]
                else:
                    forward_opposite = [forward_edge[0], -1, p3[1] - forward_edge[0] * p3[0]]
            new_p0 = line_cross_point(forward_opposite, edge)
            new_p3 = line_cross_point(forward_opposite, edge_opposite)
            fitted_parallelograms.append([new_p0, new_p1, new_p2, new_p3, new_p0])
            # or move backward edge
            new_p0 = p0
            new_p1 = p1
            new_p2 = p2
            new_p3 = p3
            new_p3 = line_cross_point(backward_edge, edge_opposite)
            if point_dist_to_line(p0, p3, p1) > point_dist_to_line(p0, p3, p2):
                # across p1
                if backward_edge[1] == 0:
                    backward_opposite = [1, 0, -p1[0]]
                else:
                    backward_opposite = [backward_edge[0], -1, p1[1] - backward_edge[0] * p1[0]]
            else:
                # across p2
                if backward_edge[1] == 0:
                    backward_opposite = [1, 0, -p2[0]]
                else:
                    backward_opposite = [backward_edge[0], -1, p2[1] - backward_edge[0] * p2[0]]
            new_p1 = line_cross_point(backward_opposite, edge)
            new_p2 = line_cross_point(backward_opposite, edge_opposite)
            fitted_parallelograms.append([new_p0, new_p1, new_p2, new_p3, new_p0])
        areas = [Polygon(t).area for t in fitted_parallelograms]
```

```
        parallelogram = np.array(fitted_parallelograms[np.argmin(areas)][:-1], dtype=np.float32)
        # sort thie polygon
        parallelogram_coord_sum = np.sum(parallelogram, axis=1)
        min_coord_idx = np.argmin(parallelogram_coord_sum)
        parallelogram = parallelogram[
            [min_coord_idx, (min_coord_idx + 1) % 4, (min_coord_idx + 2) % 4, (min_coord_idx + 3) % 4]]

        rectange = rectangle_from_parallelogram(parallelogram)
        rectange, rotate_angle = sort_rectangle(rectange)

        p0_rect, p1_rect, p2_rect, p3_rect = rectange
        for y, x in xy_in_poly:
            point = np.array([x, y], dtype=np.float32)
            # top
            geo_map[y, x, 0] = point_dist_to_line(p0_rect, p1_rect, point)
            # right
            geo_map[y, x, 1] = point_dist_to_line(p1_rect, p2_rect, point)
            # down
            geo_map[y, x, 2] = point_dist_to_line(p2_rect, p3_rect, point)
            # left
            geo_map[y, x, 3] = point_dist_to_line(p3_rect, p0_rect, point)
            # angle
            geo_map[y, x, 4] = rotate_angle
    return score_map, geo_map, training_mask


def generator(input_size=512, batch_size=32,
              background_ratio=3./8,
              random_scale=np.array([0.5, 1, 2.0, 3.0]),
              vis=False):
    image_list = np.array(get_images())
    print('{} training images in {}'.format(
        image_list.shape[0], FLAGS.training_data_path))
    index = np.arange(0, image_list.shape[0])
    while True:
        np.random.shuffle(index)
        images = []
        image_fns = []
        score_maps = []
```

```
        geo_maps = []
        training_masks = []
        for i in index:
            try:
                im_fn = image_list[i]
                im = cv2.imread(im_fn)
                # print im_fn
                h, w, _ = im.shape
                txt_fn = im_fn.replace(os.path.basename(im_fn).split('.')[1], 'txt')
                if not os.path.exists(txt_fn):
                    print('text file {} does not exists'.format(txt_fn))
                    continue

                text_polys, text_tags = load_annoataion(txt_fn)

                text_polys, text_tags = check_and_validate_polys(text_polys, text_tags, (h,
                 ↪  w))
                # if text_polys.shape[0] == 0:
                #     continue
                # random scale this image
                rd_scale = np.random.choice(random_scale)
                im = cv2.resize(im, dsize=None, fx=rd_scale, fy=rd_scale)
                text_polys *= rd_scale
                # print rd_scale
                # random crop a area from image
                if np.random.rand() < background_ratio:
                    # crop background
                    im, text_polys, text_tags = crop_area(im, text_polys, text_tags,
                     ↪  crop_background=True)
                    if text_polys.shape[0] > 0:
                        # cannot find background
                        continue
                    # pad and resize image
                    new_h, new_w, _ = im.shape
                    max_h_w_i = np.max([new_h, new_w, input_size])
                    im_padded = np.zeros((max_h_w_i, max_h_w_i, 3), dtype=np.uint8)
                    im_padded[:new_h, :new_w, :] = im.copy()
                    im = cv2.resize(im_padded, dsize=(input_size, input_size))
                    score_map = np.zeros((input_size, input_size), dtype=np.uint8)
                    geo_map_channels = 5 if FLAGS.geometry == 'RBOX' else 8
```

```python
                geo_map = np.zeros((input_size, input_size, geo_map_channels), dtype=np.float32)
                training_mask = np.ones((input_size, input_size), dtype=np.uint8)
            else:
                im, text_polys, text_tags = crop_area(im, text_polys, text_tags, crop_background=False)
                if text_polys.shape[0] == 0:
                    continue
                h, w, _ = im.shape

                # pad the image to the training input size or the longer side of image
                new_h, new_w, _ = im.shape
                max_h_w_i = np.max([new_h, new_w, input_size])
                im_padded = np.zeros((max_h_w_i, max_h_w_i, 3), dtype=np.uint8)
                im_padded[:new_h, :new_w, :] = im.copy()
                im = im_padded
                # resize the image to input size
                new_h, new_w, _ = im.shape
                resize_h = input_size
                resize_w = input_size
                im = cv2.resize(im, dsize=(resize_w, resize_h))
                resize_ratio_3_x = resize_w/float(new_w)
                resize_ratio_3_y = resize_h/float(new_h)
                text_polys[:, :, 0] *= resize_ratio_3_x
                text_polys[:, :, 1] *= resize_ratio_3_y
                new_h, new_w, _ = im.shape
                score_map, geo_map, training_mask = generate_rbox((new_h, new_w), text_polys, text_tags)

            if vis:
                fig, axs = plt.subplots(3, 2, figsize=(20, 30))
                # axs[0].imshow(im[:, :, ::-1])
                # axs[0].set_xticks([])
                # axs[0].set_yticks([])
                # for poly in text_polys:
                #     poly_h = min(abs(poly[3, 1] - poly[0, 1]), abs(poly[2, 1] - poly[1, 1]))
                #     poly_w = min(abs(poly[1, 0] - poly[0, 0]), abs(poly[2, 0] - poly[3, 0]))
                #     axs[0].add_artist(Patches.Polygon(
```

```
                    #         poly * 4, facecolor='none', edgecolor='green', linewidth=2, linestyle='-', fill=True))
                    #     axs[0].text(poly[0, 0] * 4, poly[0, 1] * 4, '{:.0f}-{:.0f}'.format(poly_h * 4, poly_w * 4),
                    #                     color='purple')
                    # axs[1].imshow(score_map)
                    # axs[1].set_xticks([])
                    # axs[1].set_yticks([])
                    axs[0, 0].imshow(im[:, :, ::-1])
                    axs[0, 0].set_xticks([])
                    axs[0, 0].set_yticks([])
                    for poly in text_polys:
                        poly_h = min(abs(poly[3, 1] - poly[0, 1]), abs(poly[2, 1] - poly[1, 1]))
                        poly_w = min(abs(poly[1, 0] - poly[0, 0]), abs(poly[2, 0] - poly[3, 0]))
                        axs[0, 0].add_artist(Patches.Polygon(
                            poly, facecolor='none', edgecolor='green', linewidth=2, linestyle='-', fill=True))
                        axs[0, 0].text(poly[0, 0], poly[0, 1], '{:.0f}-{:.0f}'.format(poly_h, poly_w), color='purple')
                    axs[0, 1].imshow(score_map[::, ::])
                    axs[0, 1].set_xticks([])
                    axs[0, 1].set_yticks([])
                    axs[1, 0].imshow(geo_map[::, ::, 0])
                    axs[1, 0].set_xticks([])
                    axs[1, 0].set_yticks([])
                    axs[1, 1].imshow(geo_map[::, ::, 1])
                    axs[1, 1].set_xticks([])
                    axs[1, 1].set_yticks([])
                    axs[2, 0].imshow(geo_map[::, ::, 2])
                    axs[2, 0].set_xticks([])
                    axs[2, 0].set_yticks([])
                    axs[2, 1].imshow(training_mask[::, ::])
                    axs[2, 1].set_xticks([])
                    axs[2, 1].set_yticks([])
                    plt.tight_layout()
                    plt.show()
                    plt.close()

                images.append(im[:, :, ::-1].astype(np.float32))
```

```
                image_fns.append(im_fn)
                score_maps.append(score_map[::4, ::4, np.newaxis].astype(np.float32))
                geo_maps.append(geo_map[::4, ::4, :].astype(np.float32))
                training_masks.append(training_mask[::4, ::4, np.newaxis].astype(np.float32))

                if len(images) == batch_size:
                    yield images, image_fns, score_maps, geo_maps, training_masks
                    images = []
                    image_fns = []
                    score_maps = []
                    geo_maps = []
                    training_masks = []
            except Exception as e:
                import traceback
                traceback.print_exc()
                continue


def get_batch(num_workers, **kwargs):
    try:
        enqueuer = GeneratorEnqueuer(generator(**kwargs), use_multiprocessing=True)
        enqueuer.start(max_queue_size=24, workers=num_workers)
        generator_output = None
        while True:
            while enqueuer.is_running():
                if not enqueuer.queue.empty():
                    generator_output = enqueuer.queue.get()
                    break
                else:
                    time.sleep(0.01)
            yield generator_output
            generator_output = None
    finally:
        if enqueuer is not None:
            enqueuer.stop()



if __name__ == '__main__':
    pass
```

Appendix C - model.py

```
import tensorflow as tf
import numpy as np
from tensorflow.contrib import slim
tf.app.flags.DEFINE_integer('text_scale', 512, '')
from nets import resnet_v1
FLAGS = tf.app.flags.FLAGS


def unpool(inputs):
    return tf.image.resize_bilinear(inputs, size=[tf.shape(inputs)[1]*2, tf.shape(inputs)[2]*2])


def mean_image_subtraction(images, means=[123.68, 116.78, 103.94]):
    '''
    image normalization
    :param images:
    :param means:
    :return:
    '''
    num_channels = images.get_shape().as_list()[-1]
    if len(means) != num_channels:
      raise ValueError('len(means) must match the number of channels')
    channels = tf.split(axis=3, num_or_size_splits=num_channels, value=images)
    for i in range(num_channels):
        channels[i] -= means[i]
    return tf.concat(axis=3, values=channels)


def model(images, weight_decay=1e-5, is_training=True):
    '''
    define the model, we use slim's implemention of resnet
    '''
    images = mean_image_subtraction(images)

    with slim.arg_scope(resnet_v1.resnet_arg_scope(weight_decay=weight_decay)):
        logits, end_points = resnet_v1.resnet_v1_50(images, is_training=is_training, scope='resnet_v1_50')
```

```python

    with tf.variable_scope('feature_fusion', values=[end_points.values]):
        batch_norm_params = {
        'decay': 0.997,
        'epsilon': 1e-5,
        'scale': True,
        'is_training': is_training
        }
        with slim.arg_scope([slim.conv2d],
                            activation_fn=tf.nn.relu,
                            normalizer_fn=slim.batch_norm,
                            normalizer_params=batch_norm_params,
                            weights_regularizer=slim.l2_regularizer(weight_decay)):
            f = [end_points['pool5'], end_points['pool4'],
                 end_points['pool3'], end_points['pool2']]
            for i in range(4):
                print('Shape of f_{} {}'.format(i, f[i].shape))
            g = [None, None, None, None]
            h = [None, None, None, None]
            num_outputs = [None, 128, 64, 32]
            for i in range(4):
                if i == 0:
                    h[i] = f[i]
                else:
                    c1_1 = slim.conv2d(tf.concat([g[i-1], f[i]], axis=-1), num_outputs[i], 1)
                    h[i] = slim.conv2d(c1_1, num_outputs[i], 3)
                if i <= 2:
                    g[i] = unpool(h[i])
                else:
                    g[i] = slim.conv2d(h[i], num_outputs[i], 3)
                print('Shape of h_{} {}, g_{} {}'.format(i, h[i].shape, i, g[i].shape))

            # here we use a slightly different way for regression part,
            # we first use a sigmoid to limit the regression range, and also
            # this is do with the angle map
            F_score = slim.conv2d(g[3], 1, 1, activation_fn=tf.nn.sigmoid,
            ↪   normalizer_fn=None)
            # 4 channel of axis aligned bbox and 1 channel rotation angle
            geo_map = slim.conv2d(g[3], 4, 1, activation_fn=tf.nn.sigmoid,
            ↪   normalizer_fn=None) * FLAGS.text_scale
```

```python
            angle_map = (slim.conv2d(g[3], 1, 1, activation_fn=tf.nn.sigmoid, normalizer_fn=None) - 0.5) * np.pi/2 # angle is between [-45, 45]
            F_geometry = tf.concat([geo_map, angle_map], axis=-1)

    return F_score, F_geometry


def dice_coefficient(y_true_cls, y_pred_cls,
                     training_mask):
    '''
    dice loss
    :param y_true_cls:
    :param y_pred_cls:
    :param training_mask:
    :return:
    '''
    eps = 1e-5
    intersection = tf.reduce_sum(y_true_cls * y_pred_cls * training_mask)
    union = tf.reduce_sum(y_true_cls * training_mask) + tf.reduce_sum(y_pred_cls * training_mask) + eps
    loss = 1. - (2 * intersection / union)
    tf.summary.scalar('classification_dice_loss', loss)
    return loss


def loss(y_true_cls, y_pred_cls,
         y_true_geo, y_pred_geo,
         training_mask):
    '''
    define the loss used for training, containing two parts,
    the first part we use dice loss instead of weighted logloss,
    the second part is the IoU loss defined in the paper
    :param y_true_cls: ground truth of text
    :param y_pred_cls: prediction of text
    :param y_true_geo: ground truth of geometry
    :param y_pred_geo: prediction of geometry
    :param training_mask: mask used in training, to ignore some text annotated by ###
    :return:
    '''
    classification_loss = dice_coefficient(y_true_cls, y_pred_cls, training_mask)
    # scale classification loss to match the iou loss part
```

```
    classification_loss *= 0.01

    # d1 -> top, d2->right, d3->bottom, d4->left
    d1_gt, d2_gt, d3_gt, d4_gt, theta_gt = tf.split(value=y_true_geo, num_or_size_splits=5,
     ↪   axis=3)
    d1_pred, d2_pred, d3_pred, d4_pred, theta_pred = tf.split(value=y_pred_geo,
     ↪   num_or_size_splits=5, axis=3)
    area_gt = (d1_gt + d3_gt) * (d2_gt + d4_gt)
    area_pred = (d1_pred + d3_pred) * (d2_pred + d4_pred)
    w_union = tf.minimum(d2_gt, d2_pred) + tf.minimum(d4_gt, d4_pred)
    h_union = tf.minimum(d1_gt, d1_pred) + tf.minimum(d3_gt, d3_pred)
    area_intersect = w_union * h_union
    area_union = area_gt + area_pred - area_intersect
    L_AABB = -tf.log((area_intersect + 1.0)/(area_union + 1.0))
    L_theta = 1 - tf.cos(theta_pred - theta_gt)
    tf.summary.scalar('geometry_AABB', tf.reduce_mean(L_AABB * y_true_cls * training_mask))
    tf.summary.scalar('geometry_theta', tf.reduce_mean(L_theta * y_true_cls * training_mask))
    L_g = L_AABB + 20 * L_theta

    return tf.reduce_mean(L_g * y_true_cls * training_mask) + classification_loss
```

Appendix D - multigpu_train.py

```python
import time
import numpy as np
import tensorflow as tf
from tensorflow.contrib import slim

tf.app.flags.DEFINE_integer('input_size', 512, '')
tf.app.flags.DEFINE_integer('batch_size_per_gpu', 14, '')
tf.app.flags.DEFINE_integer('num_readers', 16, '')
tf.app.flags.DEFINE_float('learning_rate', 0.0001, '')
tf.app.flags.DEFINE_integer('max_steps', 100000, '')
tf.app.flags.DEFINE_float('moving_average_decay', 0.997, '')
tf.app.flags.DEFINE_string('gpu_list', '1', '')
tf.app.flags.DEFINE_string('checkpoint_path', '/tmp/east_resnet_v1_50_rbox/', '')
tf.app.flags.DEFINE_boolean('restore', False, 'whether to resotre from checkpoint')
tf.app.flags.DEFINE_integer('save_checkpoint_steps', 1000, '')
tf.app.flags.DEFINE_integer('save_summary_steps', 100, '')
tf.app.flags.DEFINE_string('pretrained_model_path', None, '')

import model
import icdar

FLAGS = tf.app.flags.FLAGS

gpus = list(range(len(FLAGS.gpu_list.split(','))))


def tower_loss(images, score_maps, geo_maps, training_masks, reuse_variables=None):
    # Build inference graph
    with tf.variable_scope(tf.get_variable_scope(), reuse=reuse_variables):
        f_score, f_geometry = model.model(images, is_training=True)

    model_loss = model.loss(score_maps, f_score,
                            geo_maps, f_geometry,
                            training_masks)
    total_loss = tf.add_n([model_loss] +
     ↪  tf.get_collection(tf.GraphKeys.REGULARIZATION_LOSSES))

    # add summary
```

```python
    if reuse_variables is None:
        tf.summary.image('input', images)
        tf.summary.image('score_map', score_maps)
        tf.summary.image('score_map_pred', f_score * 255)
        tf.summary.image('geo_map_0', geo_maps[:, :, :, 0:1])
        tf.summary.image('geo_map_0_pred', f_geometry[:, :, :, 0:1])
        tf.summary.image('training_masks', training_masks)
        tf.summary.scalar('model_loss', model_loss)
        tf.summary.scalar('total_loss', total_loss)

    return total_loss, model_loss


def average_gradients(tower_grads):
    average_grads = []
    for grad_and_vars in zip(*tower_grads):
        grads = []
        for g, _ in grad_and_vars:
            expanded_g = tf.expand_dims(g, 0)
            grads.append(expanded_g)

        grad = tf.concat(grads, 0)
        grad = tf.reduce_mean(grad, 0)

        v = grad_and_vars[0][1]
        grad_and_var = (grad, v)
        average_grads.append(grad_and_var)

    return average_grads


def main(argv=None):
    import os
    os.environ['CUDA_VISIBLE_DEVICES'] = FLAGS.gpu_list
    if not tf.gfile.Exists(FLAGS.checkpoint_path):
        tf.gfile.MkDir(FLAGS.checkpoint_path)
    else:
        if not FLAGS.restore:
            tf.gfile.DeleteRecursively(FLAGS.checkpoint_path)
            tf.gfile.MkDir(FLAGS.checkpoint_path)

```

```
    input_images = tf.placeholder(tf.float32, shape=[None, None, None, 3], name='input_images')
    input_score_maps = tf.placeholder(tf.float32, shape=[None, None, None, 1], name='input_score_maps')
    if FLAGS.geometry == 'RBOX':
        input_geo_maps = tf.placeholder(tf.float32, shape=[None, None, None, 5], name='input_geo_maps')
    else:
        input_geo_maps = tf.placeholder(tf.float32, shape=[None, None, None, 8], name='input_geo_maps')
    input_training_masks = tf.placeholder(tf.float32, shape=[None, None, None, 1], name='input_training_masks')

    global_step = tf.get_variable('global_step', [], initializer=tf.constant_initializer(0), trainable=False)
    learning_rate = tf.train.exponential_decay(FLAGS.learning_rate, global_step, decay_steps=10000, decay_rate=0.94, staircase=True)
    # add summary
    tf.summary.scalar('learning_rate', learning_rate)
    opt = tf.train.AdamOptimizer(learning_rate)
    # opt = tf.train.MomentumOptimizer(learning_rate, 0.9)


    # split
    input_images_split = tf.split(input_images, len(gpus))
    input_score_maps_split = tf.split(input_score_maps, len(gpus))
    input_geo_maps_split = tf.split(input_geo_maps, len(gpus))
    input_training_masks_split = tf.split(input_training_masks, len(gpus))

    tower_grads = []
    reuse_variables = None
    for i, gpu_id in enumerate(gpus):
        with tf.device('/gpu:%d' % gpu_id):
            with tf.name_scope('model_%d' % gpu_id) as scope:
                iis = input_images_split[i]
                isms = input_score_maps_split[i]
                igms = input_geo_maps_split[i]
                itms = input_training_masks_split[i]
                total_loss, model_loss = tower_loss(iis, isms, igms, itms, reuse_variables)
                batch_norm_updates_op = tf.group(*tf.get_collection(tf.GraphKeys.UPDATE_OPS, scope))
```

```
                reuse_variables = True

                grads = opt.compute_gradients(total_loss)
                tower_grads.append(grads)

    grads = average_gradients(tower_grads)
    apply_gradient_op = opt.apply_gradients(grads, global_step=global_step)

    summary_op = tf.summary.merge_all()
    # save moving average
    variable_averages = tf.train.ExponentialMovingAverage(
        FLAGS.moving_average_decay, global_step)
    variables_averages_op = variable_averages.apply(tf.trainable_variables())
    # batch norm updates
    with tf.control_dependencies([variables_averages_op, apply_gradient_op, batch_norm_updates_op]):
        train_op = tf.no_op(name='train_op')

    saver = tf.train.Saver(tf.global_variables())
    summary_writer = tf.summary.FileWriter(FLAGS.checkpoint_path, tf.get_default_graph())

    init = tf.global_variables_initializer()

    if FLAGS.pretrained_model_path is not None:
        variable_restore_op = slim.assign_from_checkpoint_fn(FLAGS.pretrained_model_path, slim.get_trainable_variables(),
                                                             ignore_missing_vars=True)
    with tf.Session(config=tf.ConfigProto(allow_soft_placement=True)) as sess:
        if FLAGS.restore:
            print('continue training from previous checkpoint')
            ckpt = tf.train.latest_checkpoint(FLAGS.checkpoint_path)
            saver.restore(sess, ckpt)
        else:
            sess.run(init)
            if FLAGS.pretrained_model_path is not None:
                variable_restore_op(sess)

        data_generator = icdar.get_batch(num_workers=FLAGS.num_readers,
                                         input_size=FLAGS.input_size,
                                         batch_size=FLAGS.batch_size_per_gpu * len(gpus))

```

```
        start = time.time()
        for step in range(FLAGS.max_steps):
            data = next(data_generator)
            ml, tl, _ = sess.run([model_loss, total_loss, train_op], feed_dict={input_images: data[0],
                                                                                input_score_maps: data[2],
                                                                                input_geo_maps: data[3],
                                                                                input_training_masks: data[4]})
            if np.isnan(tl):
                print('Loss diverged, stop training')
                break

            if step % 10 == 0:
                avg_time_per_step = (time.time() - start)/10
                avg_examples_per_second = (10 * FLAGS.batch_size_per_gpu * len(gpus))/(time.time() - start)
                start = time.time()
                print('Step {:06d}, model loss {:.4f}, total loss {:.4f}, {:.2f} seconds/step, {:.2f} examples/second'.format(
                    step, ml, tl, avg_time_per_step, avg_examples_per_second))

            if step % FLAGS.save_checkpoint_steps == 0:
                saver.save(sess, FLAGS.checkpoint_path + 'model.ckpt', global_step=global_step)

            if step % FLAGS.save_summary_steps == 0:
                _, tl, summary_str = sess.run([train_op, total_loss, summary_op], feed_dict={input_images: data[0],
                                                                                             input_score_maps: data[2],
                                                                                             input_geo_maps: data[3],
```

```
                                                                              ↪    input_training_masks
                                                                              ↪    data[4]})
                summary_writer.add_summary(summary_str, global_step=step)

if __name__ == '__main__':
    tf.app.run()
```

Appendix E - eval.py

```
# encode=utf-8
# text detection
import time
import os
import cv2
import numpy as np
import tensorflow as tf
import model
from preproc import get_images, resize_image
from detect import detect
from recognize import recognize
import box_modifier
import enchant
import locality_aware_nms as nms_locality

tf.app.flags.DEFINE_string('test_data_path', './images/', '')
tf.app.flags.DEFINE_string('model_path', './model_CRNN/', '')
tf.app.flags.DEFINE_string('gpu_list', '0', '')
tf.app.flags.DEFINE_string('checkpoint_path', './east_icdar2015_resnet_v1_50_rbox/', '')
tf.app.flags.DEFINE_string('output_dir', './images/', '')

FLAGS = tf.app.flags.FLAGS
dictEN = enchant.Dict("en_US")
dictFR = enchant.Dict("fr_FR")


def dict_check(sim_pred):
    if dictEN.check(sim_pred) is True:
        return sim_pred + '_EN'
    elif dictFR.check(sim_pred) is True:
        return sim_pred + '_FR'


def main(argv=None):
    os.environ['CUDA_VISIBLE_DEVICES'] = FLAGS.gpu_list
    try:
        os.makedirs(FLAGS.output_dir)
    except OSError as e:
```

```
        if e.errno != 17:
            raise

    with tf.get_default_graph().as_default():
        input_images = tf.placeholder(tf.float32, shape=[None, None, None, 3], name='input_images')
        global_step = tf.get_variable('global_step', [], initializer=tf.constant_initializer(0), trainable=False)

        f_score, f_geometry = model.model(input_images, is_training=False)

        variable_averages = tf.train.ExponentialMovingAverage(0.997, global_step)
        saver = tf.train.Saver(variable_averages.variables_to_restore())

        with tf.Session(config=tf.ConfigProto(allow_soft_placement=True)) as sess:
            ckpt_state = tf.train.get_checkpoint_state(FLAGS.checkpoint_path)
            model_path = os.path.join(FLAGS.checkpoint_path, os.path.basename(ckpt_state.model_checkpoint_path))
            print('Restore from {}'.format(model_path))
            saver.restore(sess, model_path)

            im_fn_list = get_images()
            for im_fn in im_fn_list:
                im = cv2.imread(im_fn)[:, :, ::-1]
                start_time = time.time()
                im_resized, (ratio_h, ratio_w) = resize_image(im)

                timer = {'net': 0, 'restore': 0, 'nms': 0}
                start = time.time()
                score, geometry = sess.run([f_score, f_geometry], feed_dict={input_images: [im_resized]})
                timer['net'] = time.time() - start

                boxes, timer = detect(score_map=score, geo_map=geometry, timer=timer)
                print('{} : net {:.0f}ms, restore {:.0f}ms, nms {:.0f}ms'.format(
                    im_fn, timer['net'] * 1000, timer['restore'] * 1000, timer['nms'] * 1000))

                if boxes is not None:
                    boxes = boxes[:, :8].reshape((-1, 4, 2))
                    boxes[:, :, 0] /= ratio_w
```

```
                    boxes[:, :, 1] /= ratio_h

                duration = time.time() - start_time
                print('[timing] {}'.format(duration))

                text = []
                box_coor = []
                for box in boxes:
                    # to avoid submitting errors
                    box = box_modifier.sort_poly(box.astype(np.int32))
                    if np.linalg.norm(box[0] - box[1]) < 5 or np.linalg.norm(box[3] - box[0]) < 5:
                        continue
                        # cropping the boxes
                    for _ in box:
                        _box = im[(np.amin(box, 0)[1]):(np.amax(box, 0)[1]),
                               (np.amin(box, 0)[0]):(np.amax(box, 0)[0])]
                        _box_coor = [(np.amin(box, 0)[0]), (np.amax(box, 0)[0]), (np.amin(box, 0)[1]),
                                     (np.amax(box, 0)[1])]
                    if recognize(_box) is not None:
                        text.append(recognize(_box))
                        box_coor.append(_box_coor)

            matches = box_modifier.match_check(box_coor)

            for i in range(len(matches)):
                text[matches[i][0]] = text[matches[i][0]] + ' ' + text[matches[i][1]]

            box_modifier.comb_boxes(box_coor, matches)
            box_location = box_modifier.get_box_location(box_coor, im)

        return text, box_location


if __name__ == '__main__':
    tf.app.run()
```

Appendix F - preproc.py

```
import cv2
import os
import tensorflow as tf

FLAGS = tf.app.flags.FLAGS

def get_images():
    '''
    find image files in test data path
    :return: list of files found
    '''
    files = []
    exts = ['jpg', 'png', 'jpeg', 'JPG']
    for parent, dirnames, filenames in os.walk(FLAGS.test_data_path):
        for filename in filenames:
            for ext in exts:
                if filename.endswith(ext):
                    files.append(os.path.join(parent, filename))
                    break
    print('Found {} images'.format(len(files)))
    return files


def resize_image(im, max_side_len=2400):
    '''
    resize image to a size multiple of 32 which is required by the network
    :param im: the resized image
    :param max_side_len: limit of max image size to avoid out of memory in gpu
    :return: the resized image and the resize ratio
    '''
    h, w, _ = im.shape

    resize_w = w
    resize_h = h

    # limit the max side
    if max(resize_h, resize_w) > max_side_len:
```

```
        ratio = float(max_side_len) / resize_h if resize_h > resize_w else
         ↪  float(max_side_len) / resize_w
    else:
        ratio = 1.
    resize_h = int(resize_h * ratio)
    resize_w = int(resize_w * ratio)

    resize_h = resize_h if resize_h % 32 == 0 else (resize_h // 32 - 1) * 32
    resize_w = resize_w if resize_w % 32 == 0 else (resize_w // 32 - 1) * 32
    im = cv2.resize(im, (int(resize_w), int(resize_h)))

    ratio_h = resize_h / float(h)
    ratio_w = resize_w / float(w)

    return im, (ratio_h, ratio_w)
```

Appendix G - box_modifier.py

```
import itertools
import numpy as np


def sort_poly(p):
    min_axis = np.argmin(np.sum(p, axis=1))
    p = p[[min_axis, (min_axis + 1) % 4, (min_axis + 2) % 4, (min_axis + 3) % 4]]
    if abs(p[0, 0] - p[1, 0]) > abs(p[0, 1] - p[1, 1]):
        return p
    else:
        return p[[0, 3, 2, 1]]


# look for matches itertools permutations
def match_check(box_coor):
    comb_box = []
    for i, j in itertools.permutations(box_coor, 2):
        if (j[0] <= i[1] and j[2] <= i[3]) and (i[0] <= j[1] and i[2] <= j[3]):
            comb_box.append((box_coor.index(i), box_coor.index(j)))
    matches = list(set(tuple(sorted(l)) for l in comb_box))
    for k in range(len(matches)):
        if box_coor[matches[k][0]][2] <= box_coor[matches[k][1]][3]:
            matches[k] = (matches[k][1], matches[k][0])
        elif box_coor[matches[k][0]][0] <= box_coor[matches[k][1]][1]:
            matches[k] = (matches[k][1], matches[k][0])
        else:
            pass
    return matches


def comb_boxes(box_coor, matches):
    for i in matches:
        box_coor[i[0]] = [(np.minimum(box_coor[i[0]][0], box_coor[i[1]][0])),
                          (np.maximum(box_coor[i[0]][1], box_coor[i[1]][1])),
                          (np.minimum(box_coor[i[0]][2], box_coor[i[1]][2])),
                          (np.maximum(box_coor[i[0]][3], box_coor[i[1]][3]))]

        return box_coor
```

```python


def get_box_location(box_coor, im):
    im_size = np.shape(im)
    box_location = []
    for i in range(len(box_coor)):
        box_x = (box_coor[i][0] + box_coor[i][1]) / 2
        box_y = (box_coor[i][2] + box_coor[i][3]) / 2
        print(float(box_x/im_size[0]), float(box_y/im_size[1]))
        box_location.append([box_y / im_size[0], box_x / im_size[1]])

    return box_location
```

Appendix H - crnn_main.py

```
from __future__ import print_function
import argparse
import random
import torch
import torch.backends.cudnn as cudnn
import torch.optim as optim
import torch.utils.data
from torch.autograd import Variable
import numpy as np
from warpctc_pytorch import CTCLoss
import os
import utils
import dataset

import models.crnn as crnn

parser = argparse.ArgumentParser()
parser.add_argument('--trainroot', required=True, help='path to dataset')
parser.add_argument('--valroot', required=True, help='path to dataset')
parser.add_argument('--workers', type=int, help='number of data loading workers', default=2)
parser.add_argument('--batchSize', type=int, default=64, help='input batch size')
parser.add_argument('--imgH', type=int, default=32, help='the height of the input image to network')
parser.add_argument('--imgW', type=int, default=100, help='the width of the input image to network')
parser.add_argument('--nh', type=int, default=256, help='size of the lstm hidden state')
parser.add_argument('--niter', type=int, default=25, help='number of epochs to train for')
parser.add_argument('--lr', type=float, default=0.01, help='learning rate for Critic, default=0.00005')
parser.add_argument('--beta1', type=float, default=0.5, help='beta1 for adam. default=0.5')
parser.add_argument('--cuda', action='store_true', help='enables cuda')
parser.add_argument('--ngpu', type=int, default=1, help='number of GPUs to use')
parser.add_argument('--crnn', default='', help="path to crnn (to continue training)")
parser.add_argument('--alphabet', type=str, default='0123456789abcdefghijklmnopqrstuvwxyz')
parser.add_argument('--experiment', default=None, help='Where to store samples and models')
parser.add_argument('--displayInterval', type=int, default=500, help='Interval to be displayed')
```

```
parser.add_argument('--n_test_disp', type=int, default=10, help='Number of samples to display when test')
parser.add_argument('--valInterval', type=int, default=500, help='Interval to be displayed')
parser.add_argument('--saveInterval', type=int, default=500, help='Interval to be displayed')
parser.add_argument('--adam', action='store_true', help='Whether to use adam (default is rmsprop)')
parser.add_argument('--adadelta', action='store_true', help='Whether to use adadelta (default is rmsprop)')
parser.add_argument('--keep_ratio', action='store_true', help='whether to keep ratio for image resize')
parser.add_argument('--random_sample', action='store_true', help='whether to sample the dataset with random sampler')
opt = parser.parse_args()
print(opt)

if opt.experiment is None:
    opt.experiment = 'expr'
os.system('mkdir {0}'.format(opt.experiment))

opt.manualSeed = random.randint(1, 10000)  # fix seed
print("Random Seed: ", opt.manualSeed)
random.seed(opt.manualSeed)
np.random.seed(opt.manualSeed)
torch.manual_seed(opt.manualSeed)

cudnn.benchmark = True

if torch.cuda.is_available() and not opt.cuda:
    print("WARNING: You have a CUDA device, so you should probably run with --cuda")

train_dataset = dataset.lmdbDataset(root=opt.trainroot)
assert train_dataset
if not opt.random_sample:
    sampler = dataset.randomSequentialSampler(train_dataset, opt.batchSize)
else:
    sampler = None
train_loader = torch.utils.data.DataLoader(
    train_dataset, batch_size=opt.batchSize,
    shuffle=True, sampler=sampler,
    num_workers=int(opt.workers),
    collate_fn=dataset.alignCollate(imgH=opt.imgH, imgW=opt.imgW, keep_ratio=opt.keep_ratio))
```

```python
test_dataset = dataset.lmdbDataset(
    root=opt.valroot, transform=dataset.resizeNormalize((100, 32)))

nclass = len(opt.alphabet) + 1
nc = 1

converter = utils.strLabelConverter(opt.alphabet)
criterion = CTCLoss()


# custom weights initialization called on crnn
def weights_init(m):
    classname = m.__class__.__name__
    if classname.find('Conv') != -1:
        m.weight.data.normal_(0.0, 0.02)
    elif classname.find('BatchNorm') != -1:
        m.weight.data.normal_(1.0, 0.02)
        m.bias.data.fill_(0)


crnn = crnn.CRNN(opt.imgH, nc, nclass, opt.nh)
crnn.apply(weights_init)
if opt.crnn != '':
    print('loading pretrained model from %s' % opt.crnn)
    crnn.load_state_dict(torch.load(opt.crnn))
print(crnn)

image = torch.FloatTensor(opt.batchSize, 3, opt.imgH, opt.imgH)
text = torch.IntTensor(opt.batchSize * 5)
length = torch.IntTensor(opt.batchSize)

if opt.cuda:
    crnn.cuda()
    crnn = torch.nn.DataParallel(crnn, device_ids=range(opt.ngpu))
    image = image.cuda()
    criterion = criterion.cuda()

image = Variable(image)
text = Variable(text)
length = Variable(length)

```

```python
# loss averager
loss_avg = utils.averager()

# setup optimizer
if opt.adam:
    optimizer = optim.Adam(crnn.parameters(), lr=opt.lr,
                           betas=(opt.beta1, 0.999))
elif opt.adadelta:
    optimizer = optim.Adadelta(crnn.parameters(), lr=opt.lr)
else:
    optimizer = optim.RMSprop(crnn.parameters(), lr=opt.lr)


def val(net, dataset, criterion, max_iter=100):
    print('Start val')

    for p in crnn.parameters():
        p.requires_grad = False

    net.eval()
    data_loader = torch.utils.data.DataLoader(
        dataset, shuffle=True, batch_size=opt.batchSize, num_workers=int(opt.workers))
    val_iter = iter(data_loader)

    i = 0
    n_correct = 0
    loss_avg = utils.averager()

    max_iter = min(max_iter, len(data_loader))
    for i in range(max_iter):
        data = val_iter.next()
        i += 1
        cpu_images, cpu_texts = data
        batch_size = cpu_images.size(0)
        utils.loadData(image, cpu_images)
        t, l = converter.encode(cpu_texts)
        utils.loadData(text, t)
        utils.loadData(length, l)

        preds = crnn(image)
        preds_size = Variable(torch.IntTensor([preds.size(0)] * batch_size))
```

```
        cost = criterion(preds, text, preds_size, length) / batch_size
        loss_avg.add(cost)

        _, preds = preds.max(2)
        preds = preds.squeeze(2)
        preds = preds.transpose(1, 0).contiguous().view(-1)
        sim_preds = converter.decode(preds.data, preds_size.data, raw=False)
        for pred, target in zip(sim_preds, cpu_texts):
            if pred == target.lower():
                n_correct += 1

    raw_preds = converter.decode(preds.data, preds_size.data, raw=True)[:opt.n_test_disp]
    for raw_pred, pred, gt in zip(raw_preds, sim_preds, cpu_texts):
        print('%-20s => %-20s, gt: %-20s' % (raw_pred, pred, gt))

    accuracy = n_correct / float(max_iter * opt.batchSize)
    print('Test loss: %f, accuray: %f' % (loss_avg.val(), accuracy))


def trainBatch(net, criterion, optimizer):
    data = train_iter.next()
    cpu_images, cpu_texts = data
    batch_size = cpu_images.size(0)
    utils.loadData(image, cpu_images)
    t, l = converter.encode(cpu_texts)
    utils.loadData(text, t)
    utils.loadData(length, l)

    preds = crnn(image)
    preds_size = Variable(torch.IntTensor([preds.size(0)] * batch_size))
    cost = criterion(preds, text, preds_size, length) / batch_size
    crnn.zero_grad()
    cost.backward()
    optimizer.step()
    return cost


for epoch in range(opt.niter):
    train_iter = iter(train_loader)
    i = 0
    while i < len(train_loader):
```

```
        for p in crnn.parameters():
            p.requires_grad = True
        crnn.train()

        cost = trainBatch(crnn, criterion, optimizer)
        loss_avg.add(cost)
        i += 1

        if i % opt.displayInterval == 0:
            print('[%d/%d][%d/%d] Loss: %f' %
                  (epoch, opt.niter, i, len(train_loader), loss_avg.val()))
            loss_avg.reset()

        if i % opt.valInterval == 0:
            val(crnn, test_dataset, criterion)

        # do checkpointing
        if i % opt.saveInterval == 0:
            torch.save(
                crnn.state_dict(), '{0}/netCRNN_{1}_{2}.pth'.format(opt.experiment, epoch,
                 ↪   i))
```

Appendix I - adaptor.cpp

```cpp
#include "pybind11/pybind11.h"
#include "pybind11/numpy.h"
#include "pybind11/stl.h"
#include "pybind11/stl_bind.h"

#include "lanms.h"

namespace py = pybind11;


namespace lanms_adaptor {

	std::vector<std::vector<float>> polys2floats(const std::vector<lanms::Polygon> &polys) {
		std::vector<std::vector<float>> ret;
		for (size_t i = 0; i < polys.size(); i ++) {
			auto &p = polys[i];
			auto &poly = p.poly;
			ret.emplace_back(std::vector<float>{
					float(poly[0].X), float(poly[0].Y),
					float(poly[1].X), float(poly[1].Y),
					float(poly[2].X), float(poly[2].Y),
					float(poly[3].X), float(poly[3].Y),
					float(p.score),
					});
		}

		return ret;
	}


	/**
	 *
	 * \param quad_n9 an n-by-9 numpy array, where first 8 numbers denote the
	 *		quadrangle, and the last one is the score
	 * \param iou_threshold two quadrangles with iou score above this threshold
	 *		will be merged
	 *
```

```
     * \return an n-by-9 numpy array, the merged quadrangles
     */
    std::vector<std::vector<float>> merge_quadrangle_n9(
            py::array_t<float, py::array::c_style | py::array::forcecast> quad_n9,
            float iou_threshold) {
        auto pbuf = quad_n9.request();
        if (pbuf.ndim != 2 || pbuf.shape[1] != 9)
            throw std::runtime_error("quadrangles must have a shape of (n, 9)");
        auto n = pbuf.shape[0];
        auto ptr = static_cast<float *>(pbuf.ptr);
        return polys2floats(lanms::merge_quadrangle_n9(ptr, n, iou_threshold));
    }

}

PYBIND11_PLUGIN(adaptor) {
    py::module m("adaptor", "NMS");

    m.def("merge_quadrangle_n9", &lanms_adaptor::merge_quadrangle_n9,
            "merge quadrangels");

    return m.ptr();
}
```

Appendix J - recognize.py

```python
# coding=utf-8

import models.crnn as crnn
import torch
from torch.autograd import Variable
import utils
import dataset
import enchant
import unicodedata
from PIL import Image

model_path = './crrn.pth'
alphabet = '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz'
dictEN = enchant.Dict("en_US")
dictFR = enchant.Dict("fr_FR")


# https://stackoverflow.com/questions/517923/what-is-the-best-way-to-remove-accents-in-a-python-unicode-string
def remove_accents(input_str):
    nfkd_form = unicodedata.normalize('NFKD', input_str)
    only_ascii = nfkd_form.encode('ASCII', 'ignore')
    return only_ascii.decode('utf-8')


def recognize(box_coor):
    '''

    '''
    model = crnn.CRNN(32, 1, 37, 256)
    if torch.cuda.is_available():
        model = model.cuda()
    # print('loading pretrained model from %s' % model_path)
    model.load_state_dict(torch.load(model_path))

    converter = utils.strLabelConverter(alphabet)
    transformer = dataset.resizeNormalize((100, 32))

```

```
        image = Image.fromarray(box_coor).convert('L')
        image = transformer(image)
        if torch.cuda.is_available():
            image = image.cuda()
        image = image.view(1, *image.size())
        image = Variable(image)
        model.eval()
        preds = model(image)
        _, preds = preds.max(2)
        # preds = preds.squeeze(2)
        preds = preds.transpose(1, 0).contiguous().view(-1)
        preds_size = Variable(torch.IntTensor([preds.size(0)]))
        raw_pred = converter.decode(preds.data, preds_size.data, raw=True)
        sim_pred = converter.decode(preds.data, preds_size.data, raw=False)
        if len(sim_pred) > int(1):
            if dictEN.check(sim_pred) is True:
                return sim_pred
            elif dictFR.check(sim_pred) is True:
                return sim_pred
            else:
                sugg = dictFR.suggest(sim_pred)
                for i in range(0, len(sugg)):
                    sugg[i] = remove_accents(sugg[i])
                if sim_pred in sugg:
                    return sugg[0]
                else:
                    pass
        else:
            pass
```

Appendix K - crnn.py

```
import torch.nn as nn

class BidirectionalLSTM(nn.Module):

    def __init__(self, nIn, nHidden, nOut):
        super(BidirectionalLSTM, self).__init__()

        self.rnn = nn.LSTM(nIn, nHidden, bidirectional=True)
        self.embedding = nn.Linear(nHidden * 2, nOut)

    def forward(self, input):
        recurrent, _ = self.rnn(input)
        T, b, h = recurrent.size()
        t_rec = recurrent.view(T * b, h)

        output = self.embedding(t_rec)  # [T * b, nOut]
        output = output.view(T, b, -1)

        return output


class CRNN(nn.Module):

    def __init__(self, imgH, nc, nclass, nh, n_rnn=2, leakyRelu=False):
        super(CRNN, self).__init__()
        assert imgH % 16 == 0, 'imgH has to be a multiple of 16'

        ks = [3, 3, 3, 3, 3, 3, 2]
        ps = [1, 1, 1, 1, 1, 1, 0]
        ss = [1, 1, 1, 1, 1, 1, 1]
        nm = [64, 128, 256, 256, 512, 512, 512]

        cnn = nn.Sequential()

        def convRelu(i, batchNormalization=False):
            nIn = nc if i == 0 else nm[i - 1]
            nOut = nm[i]
            cnn.add_module('conv{0}'.format(i),
```

```
                        nn.Conv2d(nIn, nOut, ks[i], ss[i], ps[i]))
            if batchNormalization:
                cnn.add_module('batchnorm{0}'.format(i), nn.BatchNorm2d(nOut))
            if leakyRelu:
                cnn.add_module('relu{0}'.format(i),
                               nn.LeakyReLU(0.2, inplace=True))
            else:
                cnn.add_module('relu{0}'.format(i), nn.ReLU(True))

        convRelu(0)
        cnn.add_module('pooling{0}'.format(0), nn.MaxPool2d(2, 2))  # 64x16x64
        convRelu(1)
        cnn.add_module('pooling{0}'.format(1), nn.MaxPool2d(2, 2))  # 128x8x32
        convRelu(2, True)
        convRelu(3)
        cnn.add_module('pooling{0}'.format(2),
                       nn.MaxPool2d((2, 2), (2, 1), (0, 1)))  # 256x4x16
        convRelu(4, True)
        convRelu(5)
        cnn.add_module('pooling{0}'.format(3),
                       nn.MaxPool2d((2, 2), (2, 1), (0, 1)))  # 512x2x16
        convRelu(6, True)  # 512x1x16

        self.cnn = cnn
        self.rnn = nn.Sequential(
            BidirectionalLSTM(512, nh, nh),
            BidirectionalLSTM(nh, nh, nclass))


    def forward(self, input):
        # conv features
        conv = self.cnn(input)
        b, c, h, w = conv.size()
        assert h == 1, "the height of conv must be 1"
        conv = conv.squeeze(2)
        conv = conv.permute(2, 0, 1)  # [w, b, c]

        # rnn features
        output = self.rnn(conv)

        return output
```

Appendix L - utils.py

```python
#!/usr/bin/python
# encoding: utf-8

import torch
import torch.nn as nn
from torch.autograd import Variable
import collections


class strLabelConverter(object):
    """Convert between str and label.

    NOTE:
        Insert `blank` to the alphabet for CTC.

    Args:
        alphabet (str): set of the possible characters.
        ignore_case (bool, default=True): whether or not to ignore all of the case.
    """

    def __init__(self, alphabet, ignore_case=True):
        self._ignore_case = ignore_case
        if self._ignore_case:
            alphabet = alphabet.lower()
        self.alphabet = alphabet + '-'  # for `-1` index

        self.dict = {}
        for i, char in enumerate(alphabet):
            # NOTE: 0 is reserved for 'blank' required by wrap_ctc
            self.dict[char] = i + 1

    def encode(self, text):
        """Support batch or single str.

        Args:
            text (str or list of str): texts to convert.

        Returns:
```

```
            torch.IntTensor [length_0 + length_1 + ... length_{n - 1}]: encoded texts.
            torch.IntTensor [n]: length of each text.
        """
        if isinstance(text, str):
            text = [
                self.dict[char.lower() if self._ignore_case else char]
                for char in text
            ]
            length = [len(text)]
        elif isinstance(text, collections.Iterable):
            length = [len(s) for s in text]
            text = ''.join(text)
            text, _ = self.encode(text)
        return (torch.IntTensor(text), torch.IntTensor(length))

    def decode(self, t, length, raw=False):
        """Decode encoded texts back into strs.

        Args:
            torch.IntTensor [length_0 + length_1 + ... length_{n - 1}]: encoded texts.
            torch.IntTensor [n]: length of each text.

        Raises:
            AssertionError: when the texts and its length does not match.

        Returns:
            text (str or list of str): texts to convert.
        """
        if length.numel() == 1:
            length = length[0]
            assert t.numel() == length, "text with length: {} does not match declared length: {}".format(t.numel(), length)
            if raw:
                return ''.join([self.alphabet[i - 1] for i in t])
            else:
                char_list = []
                for i in range(length):
                    if t[i] != 0 and (not (i > 0 and t[i - 1] == t[i])):
                        char_list.append(self.alphabet[t[i] - 1])
                return ''.join(char_list)
        else:
```

```
            # batch mode
            assert t.numel() == length.sum(), "texts with length: {} does not match declared
             ↪   length: {}".format(t.numel(), length.sum())
            texts = []
            index = 0
            for i in range(length.numel()):
                l = length[i]
                texts.append(
                    self.decode(
                        t[index:index + l], torch.IntTensor([l]), raw=raw))
                index += l
            return texts


class averager(object):
    """Compute average for `torch.Variable` and `torch.Tensor`. """

    def __init__(self):
        self.reset()

    def add(self, v):
        if isinstance(v, Variable):
            count = v.data.numel()
            v = v.data.sum()
        elif isinstance(v, torch.Tensor):
            count = v.numel()
            v = v.sum()

        self.n_count += count
        self.sum += v

    def reset(self):
        self.n_count = 0
        self.sum = 0

    def val(self):
        res = 0
        if self.n_count != 0:
            res = self.sum / float(self.n_count)
        return res

```

```python

def oneHot(v, v_length, nc):
    batchSize = v_length.size(0)
    maxLength = v_length.max()
    v_onehot = torch.FloatTensor(batchSize, maxLength, nc).fill_(0)
    acc = 0
    for i in range(batchSize):
        length = v_length[i]
        label = v[acc:acc + length].view(-1, 1).long()
        v_onehot[i, :length].scatter_(1, label, 1.0)
        acc += length
    return v_onehot


def loadData(v, data):
    v.data.resize_(data.size()).copy_(data)


def prettyPrint(v):
    print('Size {0}, Type: {1}'.format(str(v.size()), v.data.type()))
    print('| Max: %f | Min: %f | Mean: %f' % (v.max().data[0], v.min().data[0],
                                              v.mean().data[0]))


def assureRatio(img):
    """Ensure imgH <= imgW."""
    b, c, h, w = img.size()
    if h > w:
        main = nn.UpsamplingBilinear2d(size=(h, h), scale_factor=None)
        img = main(img)
    return img
```

Appendix M - dataset.py

```python
#!/usr/bin/python
# encoding: utf-8

import random
import torch
from torch.utils.data import Dataset
from torch.utils.data import sampler
import torchvision.transforms as transforms
import lmdb
import six
import sys
from PIL import Image
import numpy as np


class lmdbDataset(Dataset):

    def __init__(self, root=None, transform=None, target_transform=None):
        self.env = lmdb.open(
            root,
            max_readers=1,
            readonly=True,
            lock=False,
            readahead=False,
            meminit=False)

        if not self.env:
            print('cannot creat lmdb from %s' % (root))
            sys.exit(0)

        with self.env.begin(write=False) as txn:
            nSamples = int(txn.get('num-samples'))
            self.nSamples = nSamples

        self.transform = transform
        self.target_transform = target_transform

    def __len__(self):
```

```
        return self.nSamples

    def __getitem__(self, index):
        assert index <= len(self), 'index range error'
        index += 1
        with self.env.begin(write=False) as txn:
            img_key = 'image-%09d' % index
            imgbuf = txn.get(img_key)

            buf = six.BytesIO()
            buf.write(imgbuf)
            buf.seek(0)
            try:
                img = Image.open(buf).convert('L')
            except IOError:
                print('Corrupted image for %d' % index)
                return self[index + 1]

            if self.transform is not None:
                img = self.transform(img)

            label_key = 'label-%09d' % index
            label = str(txn.get(label_key))

            if self.target_transform is not None:
                label = self.target_transform(label)

        return (img, label)


class resizeNormalize(object):

    def __init__(self, size, interpolation=Image.BILINEAR):
        self.size = size
        self.interpolation = interpolation
        self.toTensor = transforms.ToTensor()

    def __call__(self, img):
        img = img.resize(self.size, self.interpolation)
        img = self.toTensor(img)
        img.sub_(0.5).div_(0.5)
```

```
        return img


class randomSequentialSampler(sampler.Sampler):

    def __init__(self, data_source, batch_size):
        self.num_samples = len(data_source)
        self.batch_size = batch_size

    def __iter__(self):
        n_batch = len(self) // self.batch_size
        tail = len(self) % self.batch_size
        index = torch.LongTensor(len(self)).fill_(0)
        for i in range(n_batch):
            random_start = random.randint(0, len(self) - self.batch_size)
            batch_index = random_start + torch.range(0, self.batch_size - 1)
            index[i * self.batch_size:(i + 1) * self.batch_size] = batch_index
        # deal with tail
        if tail:
            random_start = random.randint(0, len(self) - self.batch_size)
            tail_index = random_start + torch.range(0, tail - 1)
            index[(i + 1) * self.batch_size:] = tail_index

        return iter(index)

    def __len__(self):
        return self.num_samples


class alignCollate(object):

    def __init__(self, imgH=32, imgW=100, keep_ratio=False, min_ratio=1):
        self.imgH = imgH
        self.imgW = imgW
        self.keep_ratio = keep_ratio
        self.min_ratio = min_ratio

    def __call__(self, batch):
        images, labels = zip(*batch)

        imgH = self.imgH
```

```
        imgW = self.imgW
        if self.keep_ratio:
            ratios = []
            for image in images:
                w, h = image.size
                ratios.append(w / float(h))
            ratios.sort()
            max_ratio = ratios[-1]
            imgW = int(np.floor(max_ratio * imgH))
            imgW = max(imgH * self.min_ratio, imgW)  # assure imgH >= imgW

        transform = resizeNormalize((imgW, imgH))
        images = [transform(image) for image in images]
        images = torch.cat([t.unsqueeze(0) for t in images], 0)

        return images, labels
```

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467, 2016.

[2] Ouais Alsharif and Joelle Pineau. End-to-end text recognition with hybrid HMM maxout models. *CoRR*, abs/1310.1811, 2013.

[3] B. Bai, F. Yin, and C. L. Liu. A fast stroke-based method for text detection in video. In *2012 10th IAPR International Workshop on Document Analysis Systems*, pages 69–73, March 2012.

[4] A. Bissacco, M. Cummins, Y. Netzer, and H. Neven. Photoocr: Reading text in uncontrolled conditions. In *2013 IEEE International Conference on Computer Vision*, pages 785–792, Dec 2013.

[5] Jeffrey R. Blum, Mathieu Bouchard, and Jeremy R. Cooperstock. Whats around me? spatialized audio augmented reality for blind users with a smartphone.

[6] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, Nov 1986.

[7] H. Chen, S. S. Tsai, G. Schroth, D. M. Chen, R. Grzeszczuk, and B. Girod. Robust text detection in natural images with edge-enhanced maximally stable extremal regions. In *2011 18th IEEE International Conference on Image Processing*, pages 2609–2612, Sept 2011.

[8] Xiangrong Chen and A. L. Yuille. Detecting and reading text in natural scenes. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 2, pages II–366–II–373 Vol.2, June 2004.

[9] Adam Coates, Blake Carpenter, Sanjeev Satheesh, Bipin Suresh, Tao Wang, David J. Wu, and Andrew Y. Ng. Text detection and character recognition in scene images with unsupervised feature learning.

[10] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

[11] M. Donoser, H. Bischof, and S. Wagner. Using web search engines to improve text recognition. In *2008 19th International Conference on Pattern Recognition*, pages 1–4, Dec 2008.

[12] Tefilo Emdio de Campos, Bodla Rakesh Babu, and Manik Varma. Character recognition in natural images. 2:273–280, 01 2009.

[13] B. Epshtein, E. Ofek, and Y. Wexler. Detecting text in natural scenes with stroke width transform. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2963–2970, June 2010.

[14] Alex Graves, Santiago Fernndez, and Faustino Gomez. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *In Proceedings of the International Conference on Machine Learning, ICML 2006*, pages 369–376, 2006.

[15] S. M. Hanif and L. Prevost. Text detection and localization in complex scene images using constrained adaboost algorithm. In *2009 10th International Conference on Document Analysis and Recognition*, pages 1–5, July 2009.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[17] Pan He, Weilin Huang, Yu Qiao, Chen Change Loy, and Xiaoou Tang. Reading scene text in deep convolutional sequences. 06 2015.

[18] Sepp Hochreiter and Jrgen Schmidhuber. Long short-term memory, 1997.

[19] Sanghoon Hong, Byung-Seok Roh, Kye-Hyeon Kim, Yeongjae Cheon, and Minje Park. Pvanet: Lightweight deep neural networks for real-time object detection. *CoRR*, abs/1611.08588, 2016.

[20] Jan Hendrik Hosang, Rodrigo Benenson, and Bernt Schiele. Learning non-maximum suppression. *CoRR*, abs/1705.02950, 2017.

[21] Lichao Huang, Yi Yang, Yafeng Deng, and Yinan Yu. Densebox: Unifying landmark localization with end to end object detection. *CoRR*, abs/1509.04874, 2015.

[22] R. Huang, P. Shivakumara, and S. Uchida. Scene character detection by an edge-ray filter. In *2013 12th International Conference on Document Analysis and Recognition*, pages 462–466, Aug 2013.

[23] M. R. Islam, C. Mondal, M. K. Azam, and A. S. M. J. Islam. Text detection and recognition using enhanced mser detection and a novel ocr technique. In *2016 5th International Conference on Informatics, Electronics and Vision (ICIEV)*, pages 15–20, May 2016.

[24] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman. Synthetic data and artificial neural networks for natural scene text recognition. In *Workshop on Deep Learning, NIPS*, 2014.

[25] Vedaldi A. Jaderberg M. and Zisserman A. Deep feature for text spotting. pages 512–528, 2014.

[26] Yingying Jiang, Xiangyu Zhu, Xiaobing Wang, Shuli Yang, Wei Li, Hua Wang, Pei Fu, and Zhenbo Luo. R2CNN: rotational region CNN for orientation robust scene text detection. *CoRR*, abs/1706.09579, 2017.

[27] D. Karatzas, L. Gomez-Bigorda, A. Nicolaou, S. Ghosh, A. Bagdanov, M. Iwamura, J. Matas, L. Neumann, V. R. Chandrasekhar, S. Lu, F. Shafait, S. Uchida, and E. Valveny. Icdar 2015 competition on robust reading. In *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*, pages 1156–1160, Aug 2015.

[28] Kwang In Kim, Keechul Jung, and Jin Hyung Kim. Texture-based approach for text detection in images using support vector machines and continuously adaptive mean shift algorithm. volume 25, pages 1631–1639, Dec 2003.

[29] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[30] Tao Kong, Anbang Yao, Yurong Chen, and Fuchun Sun. Hypernet: Towards accurate region proposal generation and joint object detection. *CoRR*, abs/1604.00600, 2016.

[31] J. J. Lee, P. H. Lee, S. W. Lee, A. Yuille, and C. Koch. Adaboost for text detection in natural scene. In *2011 International Conference on Document Analysis and Recognition*, pages 429–434, Sept 2011.

[32] Minhua Li and Chunheng Wang. An adaptive text detection approach in images and video frames. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 72–77, June 2008.

[33] Jian Liang, David S. Doermann, and Huiping Li. Camera-based analysis of text and documents: a survey. *International Journal of Document Analysis and Recognition (IJDAR)*, 7:84–104, 2004.

[34] M. Liao, B. Shi, and X. Bai. TextBoxes++: A Single-Shot Oriented Scene Text Detector. *ArXiv e-prints*, January 2018.

[35] Minghui Liao, Baoguang Shi, Xiang Bai, Xinggang Wang, and Wenyu Liu. Textboxes: A fast text detector with a single deep neural network. *CoRR*, abs/1611.06779, 2016.

[36] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[37] X. Liu, D. Liang, S. Yan, D. Chen, Y. Qiao, and J. Yan. FOTS: Fast Oriented Text Spotting with a Unified Network. *ArXiv e-prints*, January 2018.

[38] X. Liu and W. Wang. Robustly extracting captions in videos based on stroke-like edges and spatio-temporal analysis. *IEEE Transactions on Multimedia*, 14(2):482–489, April 2012.

[39] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide baseline stereo from maximally stable extremal regions. In *In Proc. BMVC*, pages 384–393, 2002.

[40] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool. A comparison of affine region detectors. *International Journal of Computer Vision*, 65(1):43–72, Nov 2005.

[41] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. *CoRR*, abs/1606.04797, 2016.

[42] A. Mishra, K. Alahari, and C. V. Jawahar. Scene text recognition using higher order language priors. In *BMVC*, 2012.

[43] Anand Mishra, Karteek Alahari, and C V. Jawahar. Scene text recognition using higher order language priors. 09 2012.

[44] L. Neumann and J. Matas. Real-time scene text localization and recognition. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3538–3545, June 2012.

[45] Tatiana Novikova, Olga Barinova, Pushmeet Kohli, and Victor Lempitsky. Large-lexicon attribute-consistent text recognition in natural images.

[46] Y. F. Pan, X. Hou, and C. L. Liu. A hybrid approach to detect and localize texts in natural scene images. *IEEE Transactions on Image Processing*, 20(3):800–813, March 2011.

[47] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[48] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[49] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[50] Zohra Saidane and Christophe Garcia. Automatic scene text recognition using a convolutional neural network. In *In Proceedings of the Second International Workshop on Camera-Based Document Analysis and Recognition (CBDAR*, 2007.

[51] Baoguang Shi, Xiang Bai, and Cong Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *CoRR*, abs/1507.05717, 2015.

[52] Baoguang Shi, Xinggang Wang, Pengyuan Lv, Cong Yao, and Xiang Bai. Robust scene text recognition with automatic rectification. *CoRR*, abs/1603.03915, 2016.

[53] Cunzhao Shi, Chunheng Wang, Baihua Xiao, Yang Zhang, Song Gao, and Zhong Zhang. Scene text recognition using part-based tree-structured character detection. pages 2961–2968, 06 2013.

[54] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[55] A. Tabassum and S. A. Dhondse. Text detection using mser and stroke width transform. In *2015 Fifth International Conference on Communication Systems and Network Technologies*, pages 568–571, April 2015.

[56] Zhi Tian, Weilin Huang, He Tong, Pan He, and Yu Qiao. Detecting text in natural image with connectionist text proposal network. volume 9912, pages 56–72, 10 2016.

[57] Andreas Veit, Tomas Matera, Lukas Neumann, Jiri Matas, and Serge J. Belongie. Coco-text: Dataset and benchmark for text detection and recognition in natural images. *CoRR*, abs/1601.07140, 2016.

[58] Kai Wang, B. Babenko, and S. Belongie. End-to-end scene text recognition. In *2011 International Conference on Computer Vision*, pages 1457–1464, Nov 2011.

[59] T. Wang, D. J. Wu, A. Coates, and A. Y. Ng. End-to-end text recognition with convolutional neural networks. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 3304–3308, Nov 2012.

[60] Tao Wang, David J Wu, Adam Coates, and Andrew Y Ng. End-to-end text recognition with convolutional neural networks. 02 2018.

[61] W. Wu, D. Chen, and J. Yang. Integrating co-training and recognition for text detection. In *2005 IEEE International Conference on Multimedia and Expo*, pages 4 pp.–, July 2005.

[62] C. Yao, X. Bai, W. Liu, Y. Ma, and Z. Tu. Detecting texts of arbitrary orientations in natural images. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1083–1090, June 2012.

[63] Cong Yao, Xiang Bai, Nong Sang, Xinyu Zhou, Shuchang Zhou, and Zhimin Cao. Scene text detection via holistic, multi-channel prediction. *CoRR*, abs/1606.09002, 2016.

[64] Q. Ye and D. Doermann. Text detection and recognition in imagery: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(7):1480–1500, 2015.

[65] C. Yi and Y. Tian. Localizing text in scene images by boundary clustering, stroke segmentation, and string fragment classification. *IEEE Transactions on Image Processing*, 21(9):4256–4268, Sept 2012.

[66] X. C. Yin, Z. Y. Zuo, S. Tian, and C. L. Liu. Text detection, tracking and recognition in video: A comprehensive survey. *IEEE Transactions on Image Processing*, 25(6):2752–2773, June 2016.

[67] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

[68] Zheng Zhang, Zhang Chengquan, Wei Shen, Cong Yao, Wenyu Liu, and Xiang Bai. Multi-oriented text detection with fully convolutional networks. pages 4159–4167, 06 2016.

[69] X. Zhou, C. Yao, H. Wen, Y. Wang, S. Zhou, W. He, and J. Liang. East: An efficient and accurate scene text detector. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2642–2651, July 2017.

[70] Karel Zimmermann. A new class of learnable detectors: Cser- class specific extremal regions.

List of Abbreviations

IEEE: Institute of Electrical and Electronics Engineers, Inc.

WHO: World Health Organization

CVPR: The Conference on Computer Vision and Pattern Recognition

ICCV: International Conference on Computer Vision

ICDAR: International Conference on Document Analysis and Recognition

OCR: Optical Character Recognition

CC: Connected Component

MSER: Maximally Stable Extremal Regions

SWT: Stroke Width Transform

CRF: Conditional Random Field

NN: Neural Network

DNN: Deep Neural Network

CNN: Convolutional Neural Network

DCNN: Deep Convolutional Neural Network

FCN: Fully Convolutional Network

RNN: Recurrent Neural Network

EAST: Easy Accurate Scene Text

RCNN & CRNN: Convolutional Recurrent Neural Network

NMS: Non Maximum Suppression

LSTM: Long Short Term Memory

CTC: Connectionist Temporal Classification

GPU: Graphics Processing Unit

CPU: Central Processing Unit

GHz: Gigahertz, 1 billion hertz

SDK: Software Development Kit

JSON: JavaScript Object Notation

GPS: Global Positioning System