FESTIV: Fast Extraction of Scene Text in Video

by

Marshall Wice

A thesis
presented to McMaster University
in fulfillment of the
thesis requirement for the degree of
Master of Engineering
in
Computing and Software

Hamilton, Ontario, Canada, 2018

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

In this project we have designed an effective pipeline for quickly extracting scene text from videos. Depthwise Separable Convolutions (DSCs) are used in the feature extraction portions of the text detection and recognition algorithms, in order to reduce inference time and model size. Sparse optical flow is further used to accelerate inference time by tracking the location of text across video frames. Finally, a probabilistic language model is used to correct erroneous predictions. On a home video, the pipeline is capable of achieving an *F-measure* of 0.38 on 512×512 RGB frames with an average inference time of 0.82fps, representing an increase in accuracy of 10.6% and an increase in speed of 212.5% over its single image baseline. Altogether, the model takes 126.4Mb of disk space, making it small enough to fit onto a mobile device.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr. Rong Zheng, for her guidance and allowing me to explore the topic of my interests.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Extracting, understanding, and using text signage found in natural scenes is an important task because it provides solutions to a variety of problems such as text-based geolocation, multi-language translation, image retrieval, and autonomous vehicle navigation. Moreover, text extraction from video can be more beneficial than from a static image. This is because the redundant information present in videos can be leveraged in order to create higher performing solutions. Despite this need for fast, accurate, and robust algorithms, subproblems of text extraction (text detection, recognition, and tracking) from video have largely been unresolved and remain active research topics. Furthermore, to the best of our knowledge there exists no pipeline for extracting text from videos in real-time on embedded and mobile devices.

The two primary components in text extraction are text detection and text recognition. Text detection is the process of taking a scene image as input and returning the regions of the image which contain text. These regions are typically defined by oriented boxes or quadrilaterals, called bounding boxes. Text recognition is the process of taking the cropped bounding boxes and interpreting them as a sequence of characters.

In this project we propose a pipeline consisting of the SegLink [41] algorithm for text detection, the Convolutional Recurrent Neural Network (CRNN) [42] algorithm for text recognition, and sparse optical flow algorithms [43, 30] for text tracking. The accuracy of the text extraction is further enhanced by utilizing a probabilistic language model to correct erroneous text predictions.

In Chapter 2 we review state-of-the-art research in the fields of object detection, text detection, text recognition, object tracking, and text tracking. In Chapter 3, a high level

explanation of the proposed pipeline is given, followed by in-depth explanations of each pipeline component. In Chapter 4, the pipeline components are evaluated individually and as a whole, against other methods. Finally, in Chapter 5 we give concluding thoughts and discuss opportunities for future work.

Chapter 2

Literature Review

2.1 Object Detection and Classification

Most deep-learning based general object detection algorithms can roughly be categorized as either region-proposal based or single-shot. A region-proposal based algorithm consists of two main steps. The first is to generate a fixed number of interesting regions in an image which are likely to contain objects. In R-CNN [14] and Fast R-CNN [13] a selective search algorithm is used to produce region proposals by comparing the colour, intensity, and textures of pixels with neighbouring pixels. In Faster R-CNN [36], a Region Proposal Network (RPN) learns to generate the regions which are associated with fixed-size reference bounding boxes. In the second step, a general purpose Convolutional Neural Network (CNN) is run on the proposed regions in order to extract deep features and make class predictions. Finally, linear regression is used to tighten bounding box cordinates. In contrast, a single-shot based detection algorithm directly makes class predictions and produces object bounding boxes in a single step. For example, YOLO [35] divides an input image into a fixed grid and for each grid space generates a fixed number of bounding boxes as well as class probabilities for the boxes. SSD [27] improves on YOLO by making predictions on multiple feature maps of different sizes rather than only on the final feature map. Although both region-proposal and single-shot detection methods are capable of achieving high accuracies on common object detection benchmarks, single-shot methods are often faster due to their lack of a region proposal phase.

2.2 Text Detection

Text detection is a special case of general object detection. Although general object detection algorithms can be used for scene text detection, they often perform poorly due to the extreme aspect ratios and orientation of text. For this reason researchers have developed text-specific detection algorithms which can be categorized by the strategy they employ to localize text: character-based, word-based, or text-line/shape based.

Character-based: Maximally Stable Extremal Regions (MSERs) [31] have been used in order to find regions of an image that likely contain characters [24]. Optical Character Recognition (OCR) is then used to recognize characters from the region. Finally, a Markov Random Field (MRF) model is used to combine extracted characters into words.

Word-based: SegLink [41] jointly predicts text parts and their relationship with one another using an SSD based network. TextBoxes++ [25] directly localizes text and produces confidence scores for the predictions using an SSD based network. EAST [50] uses a U-shaped, fully convolutional network to directly predict text/non-text scores and text locations. FCRN [15] uses a fully convolutional network to detect horizontal text regions, and then uses regression to tighten the bounds. Based on the Faster R-CNN [36] architecture, R2CNN [20] employs an RPN to generate proposals, and then applies an inclined Non-Maximum Suppression (NMS) to regress the predictions.

Text-line/Shape-based: CTPN [47] densely slides a window over feature maps to detect nearly horizontal text parts which are then chained together with a Recurrent Neural Network (RNN).

2.3 Text Recognition

Text recognition often follows text detection in order to extract the characters in the detected text regions. Similar to text detection, text recognition algorithms can either be character-based or word-based, but not text-line/shape based. Character-based algorithms are essentially multiclass-classifiers which take a single character image as input and produce a prediction on which character is shown. This problem is known as Optical Character Recognition (OCR) and already has documented solutions with accuracies in the 99th percentile. While fundamentally less difficult than its text detection counterpart, word-based text recognition algorithms remain active research topics. Recent attempts at solving word-based text recognition have predominantly involved the combination of utilizing CNNs to extract deep features of the text and then using either sliding windows

or Recurrent Neural Networks (RNNs) to make individual character predictions, which are then combined into words. CRNN [42] uses a CNN to extract character features and passes these to an RNN and Connectionist Temporal Classification (CTC) to yield character predictions. Conversely, other methods densely slide windows at different scales over the input image. For each window, a CNN is applied to extract deep features which are then chained together with an RNN [49] to produce word predictions. Another method uses a Spatial Transformer Network (STN) is used to straighten curved text images before feeding them into CRNN [48].

2.4 Text Extarction

Recently, some research has focused on single networks which can directly extract text from scene images. The primary advantage of these networks is that they provide for a trainable way to represent bounding boxes. Deep TextSpotter [5], a word-based algorithm, proposes an end-to-end text detection and recognition network using the YOLO [35] architecture, an RPN, and a Connectionist Temporal Classification (CTC) network for recognition. STN-OCR [2] jointly detects and recognizes individual characters from an input image using a Spatial Transformer Network (STN) for character localization, and OCR for recognition.

When deciding which of the aforementioned text detection, recognition, and extraction algorithms to use for this project, we first dismissed any algorithm which was incapable of detecting oriented text, since that was a required feature. Taking into consideration the performance and inference time of the remaining methods (shown in Tables 2.1, 2.2, and 2.3), we chose to use the SegLink [41] algorithm for text detection and the CRNN [42] algorithm for text recognition. Additionally, the SegLink text detection algorithm has the ability to detect text of any orientation and length.

2.5 Object Tracking and Detection in Videos

Video is a sequence of still images which are played back at a high speed, and thus contains more information than a single image. Using this additional information can increase the accuracy of detection and recognition algorithms, reduce the computation through tracking, or both. Deep Feature Flow [53] presents a video object detection framework

[1]Although this method reports 500fps, it should theoretically be slower than CRNN which reports 6.25fps since it is the same algorithm with additional preprocessing components

Table 2.1: Comparison of text detection algorithms on the ICDAR 2013 and ICDAR 2015 benchmark datasets

	ICDAR 2013		ICDAR 2015	
Method	F-measure	FPS	F-measure	FPS
TextBoxes++	0.88	-	0.82	11.6
EAST	-	-	0.78	13.2
SegLink	0.85	20.6	0.75	-
R2CNN	0.88	-	0.75	2.2
FCRN	0.83	-	-	-
CTPN	0.88	7.14	0.61	-
MSER+OCR [24]	-	-	-	-

Table 2.2: Results of text recognition algorithms on the ICDAR 2013 dataset

	ICDAR 2013	
Method	Accuracy	FPS
CRNN	0.87	6.25
CRNN with STN	0.89	500[1]
Sliding CNNs [49]	0.85	66.66

in which a feature extractor is run only on sparse keyframes. An optical flow network is further used to propagate feature maps from keyframes to non-keyframes, which are used for object detection. FGFA [52] works similarly to Deep Feature Flow with the exception that all frames are considered keyframes. Feature maps are propagated to the next keyframe and then combined with the current frames feature maps for enhanced detection. Both Deep Feature Flow and FGFA, align propagated features using the deep optical flow estimation network FlowNet [12]. Other research [6, 51], combines the ideas from Deep Feature Flow and FGFA to achieve better accuracy and inference time. A three-dimensional SSD framework is used for action recognition but also applies to tracking in [21]. Other works [26, 8], combine the single image SSD framework with RNNs to capture temporal information. STSN [3] learns to spatially sample features from previous video frames using Deformable Convolutions Networks (DCNs) [10]. ROLO [33] combines YOLO with RNNs to create a fast object tracker. The article pertaining to [37] uses SIFT features [29] and Fisher Vectors [38] for text tracking to improve OCR accuracy.

Table 2.3: End-to-end results of text extraction algorithms on the ICDAR 2013 and ICDAR 2015 datasets

	ICDAR 2013		ICDAR 2015	
Method	F-measure	FPS	F-measure	FPS
Deep TextSpotter	0.77	10.0	0.47	9.0
STN-OCR	-	-	-	-

Due to time constraints of this project, basic optical flow algorithms [43, 30] are utilized for text tracking. Future work should expand the pipeline to include more advanced tracking mechanisms such as FGFA.

Chapter 3

Methodology

3.1 Solution Overview

The text extraction pipeline used in this project consists of four components: a text detection algorithm, a text recognition algorithm, a tracking algorithm, and a language model. The goal of this section is to describe how each of these components come together to achieve text extraction. Figure 3.1 depicts the entire pipeline where $frame_i$ is the i-th video frame, $text_i$ is the i-th extracted text, N_{det} is the text detection network, N_{rec} is the text recognition network, *optical flow* is the tracking algorithm, and *language model* is the language model. Arrows represent the flow of data from one component to another.

In this pipeline we let every 3rd frame be a keyframe and all other frames be non-key frames. If the current frame is a key frame, it is fed into a text detection network N_{det}. This network outputs the oriented bounding boxes of all text found within the frame. If the current frame is a non-key frame, it is fed into an optical flow tracking algorithm along with the previous frame and the previously known bounding boxes. This has the effect of reducing the inference time of the pipeline as optical flow tracking is nearly three times faster than performing text detection. The tracking algorithm outputs the new bounding boxes locations. For both key frames and non-key frames, each of the images within the bounding boxes are fed into a text recognition network N_{rec} which outputs predicted character sequences (words). The predicted character sequences are processed by the language model in order to correct any erroneous character predictions. Furthermore, previous word predictions are utilized in forward to the language model to improve predictions.

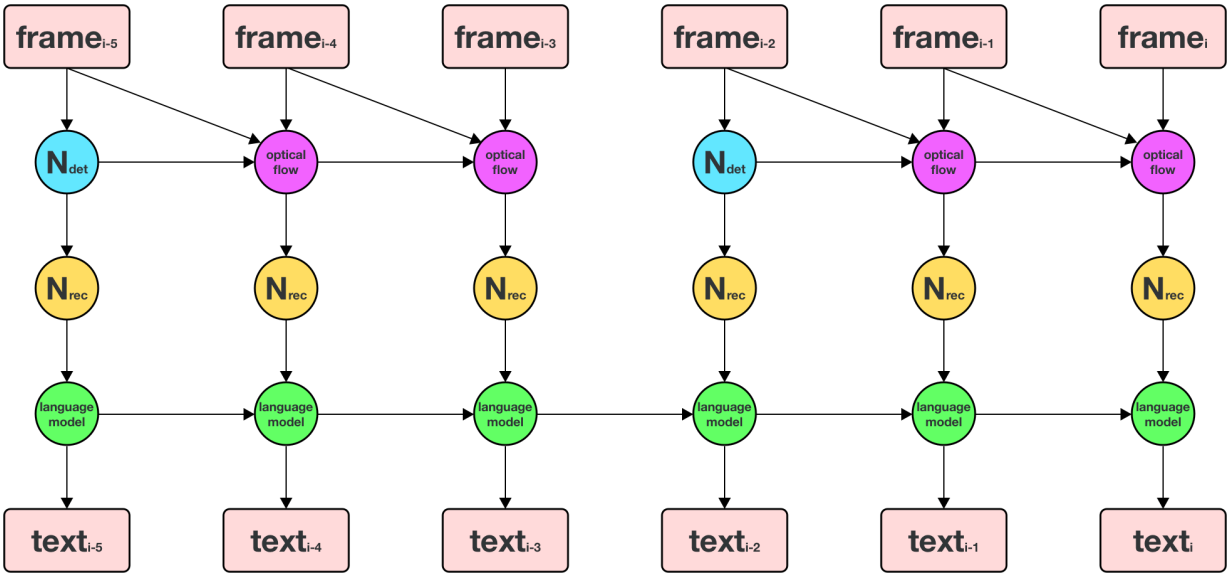

Figure 3.1: *An overview of the proposed pipeline*

3.2 Depthwise Separable Convolutions

Depthwise Separable Convolutions (DSCs) [44] are a variant of standard convolutions in which the process of filtering and combining features are split into two separate steps rather than performed simultaneously as in standard convolutions. In the first step, a single kernel is applied to each input channel, which we call the depthwise convolution. Then a 1×1 pointwise kernel is applied to combine the outputs of the depthwise convolution into the final output volume. This small change in the way we convolve over the input has the effect of drastically reducing the cost of performing the operation and the overall model size. Furthermore, MobileNets [17] showed that using DSCs instead of standard convolutions has little impact on object detection algorithm accuracies.

3.2.1 The Standard Convolution

To understand how DSC improves on the standard convolution, we first need to understand how a standard convolution is applied, and what the cost of this operation is. Consider an input volume F with dimensions $D_F \times D_F \times M$, where D_F is the width and height of the input[1] and M is the depth or number of channels. For instance, if this input volume was

an RGB image, M equals 3.

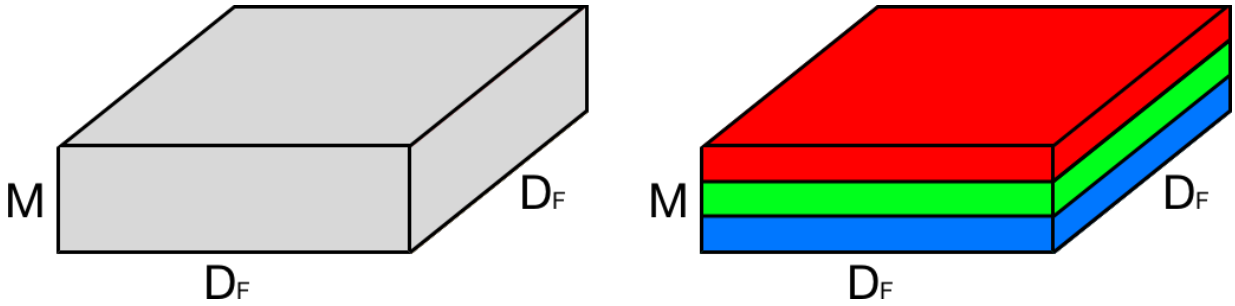

Figure 3.2: *(Left) Input volume F. (Right) Visualization of input volume F as RGB image*

Now consider a kernel (filter) of shape $D_K \times D_K \times M$. Passing this kernel over the entire input volume would yield an output volume G, with dimensions $D_G \times D_G \times 1$ where D_G is a function of the input size D_F, the kernel size D_K, the stride S, and the padding P. Specifically, $D_G = ((D_F - D_K + 2P)/S) + 1$. If we apply N kernels (filters), the output volume G will therefore have shape $D_G \times D_G \times N$.

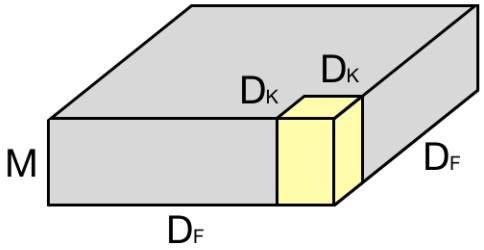

Figure 3.3: *Input volume F in gray, and kernel K in yellow*

3.2.2 Cost of the Standard Convolution

In analyzing the cost of the standard convolution, we first must determine the best way to measure cost. Typically, the cost of a convolution is measured by the number of multiplications that must be made in order to complete the operation. This is because out of the two sub-operations which take place in a convolution (multiplication and addition), multiplication is the most costly.

[1]When the width and height of the input are different, a similar analysis applies.

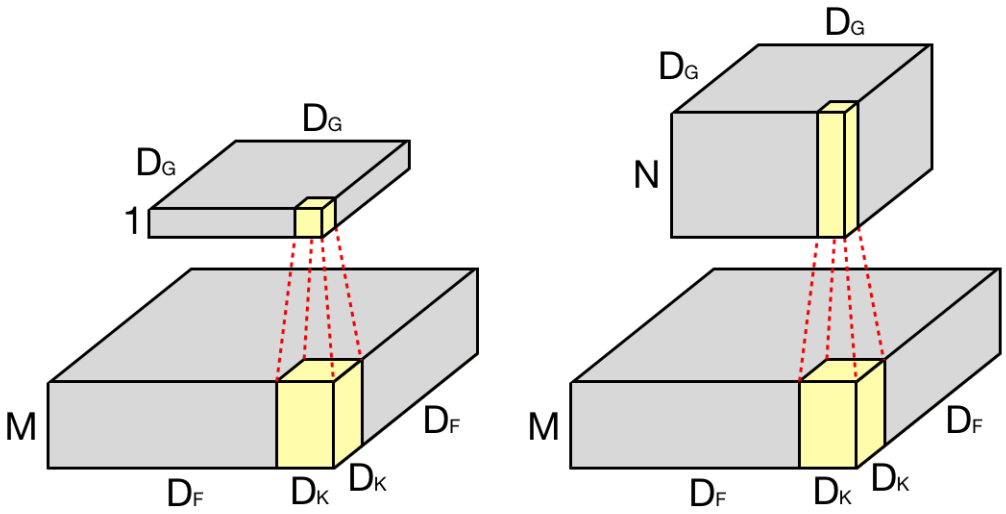

Figure 3.4: *(Left) Activation and mapping of a single kernel to its output volume G. (Right) Activation and mapping of N kernels to the output volume G*

The number of multiplications that happen when a kernel of shape $D_K \times D_K \times M$ is applied to a region of the input volume with shape $D_K \times D_K \times M$ is of course $D_K \times D_K \times M$. Sliding the kernel over the entire input volume would therefore require $D_K \times D_K \times M \times D_G \times D_G$ multiplications. Finally, with N kernels to apply, we need to perform $D_K \times D_K \times M \times D_G \times D_G \times N$ multiplications in total.

3.2.3 Depthwise Separable Convolution

Depthwise Separable Convolutions are made up of two layers, depthwise convolutions and pointwise convolutions. We use depthwise convolutions to apply a single filter per each input channel (input depth). Pointwise convolution, which is a 1×1 convolution, is then used to create a linear combination of the depthwise layer output.

When referring to depthwise convolution, let us consider the same input volume F as before, with dimensions $D_F \times D_F \times M$, where D_F is the width and height of the input, and M is the depth or number of channels. Furthermore, consider a kernel of shape $D_K \times D_K \times 1$ instead of the traditional kernel with shape $D_K \times D_K \times M$. That is, this kernel will only be applied to a single channel of the input volume. Passing this kernel over the entire channel will yield an output volume of shape $D_G \times D_G \times 1$. Since this kernel only slides

over one channel, we must pass M such kernels over the respective channel of the input volume in order to cover the entirety of F. This results in an output volume G' of shape $D_G \times D_G \times M$.

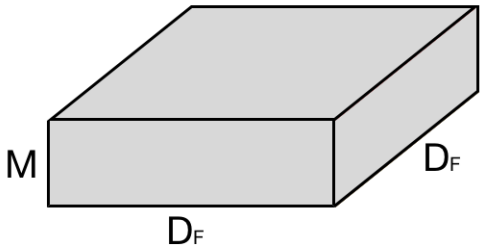

Figure 3.5: *Input volume F*

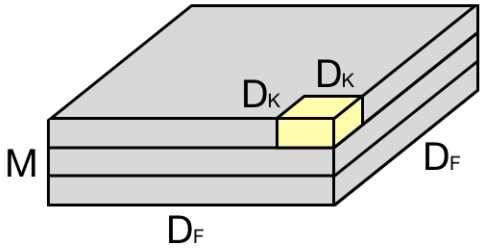

Figure 3.6: *Input volume F in gray and single channel kernel K in yellow*

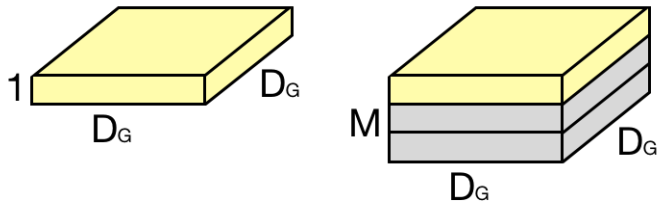

Figure 3.7: *(Left) Output volume G' from sliding the kernel K over the entire channel. (Right) Output volume G' from sliding M kernels over M channels*

After completing the depthwise convolution (filtering) layer, we are ready for pointwise convolution (combining). In pointwise convolution, the input volume is the shape G' with shape $D_G \times D_G \times M$ (the output from the depthwise convolution layer). In pointwise

convolution, the kernel takes the shape $1 \times 1 \times M$ so that it acts as a linear combination of the M channels. When such a kernel is passed over the entire input, it yields an output shape of size $D_G \times D_G \times 1$ (the output has the same width and height as the input). Furthermore, if we apply N kernels over the input, we get a final output volume G with dimensions $D_G \times D_G \times N$. Notice that this is the exact same shape as we received with standard convolution.

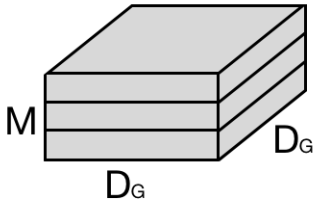

Figure 3.8: *Input volume G' to the pointwise convolution (the output from the depthwise convolution)*

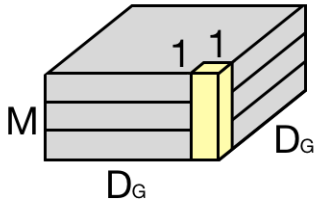

Figure 3.9: *Input volume G' in gray with linear kernel K in yellow*

3.2.4 Cost of the Depthwise Separable Convolution

The cost of DSC is the number of multiplications in a depthwise convolution plus the number of multiplications in a pointwise convolution. The number of multiplications that happen when a kernel of shape $D_K \times D_K \times 1$ is applied to a region of the input volume with shape $D_K \times D_K \times 1$, is of course $D_K \times D_K \times 1$. Sliding the kernel over the entire channel would therefore require $D_K \times D_K \times 1 \times D_G \times D_G$ multiplications. If we do this over all M channels, we would have performed $D_K \times D_K \times 1 \times D_G \times D_G \times M$ multiplications in total. Likewise, for pointwise convolution, if we apply a kernel of shape $1 \times 1 \times M$ to a patch of shape $1 \times 1 \times M$, we are performing $1 \times 1 \times M$ multiplications. Doing this over

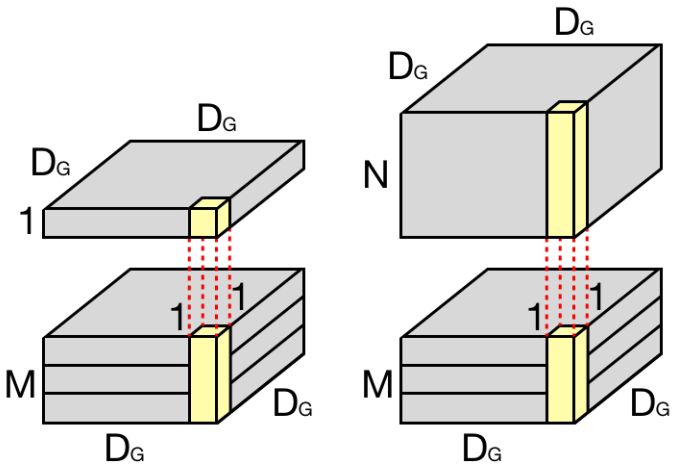

Figure 3.10: *(Left) Activation and mapping of a single kernel to its output volume G. (Right) Activation and mapping of N kernels to the output volume G*

the entire input would therefore result in $1 \times 1 \times M \times D_G \times D_G$ multiplications. If we apply N such kernels, this would cost us $1 \times 1 \times M \times D_G \times D_G \times N$ multiplications for the pointwise convolution step. Therefore, the total multiplications of a DSC is the number of multiplications in the depthwise convolution plus the number of multiplications in the pointwise convolution is $D_K \times D_K \times 1 \times D_G \times D_G \times M + 1 \times 1 \times M \times D_G \times D_G \times N$.

Finally, the cost reduction from using DSCs instead of standard convolutions is:

$$\frac{D_K \times D_K \times M \times D_G \times D_G \times N}{D_K \times D_K \times 1 \times D_G \times D_G \times M + 1 \times 1 \times M \times D_G \times D_G \times N} = \frac{1}{N} + \frac{1}{{D_K}^2}$$

To appreciate how much more efficient DSCs are over standard convolutions, consider the following values. If we have a standard convolution with $N = 1024$ kernels and kernel size $D_K = 3$, the cost reduction of switching to DSC would be:

$$\frac{1}{N} + \frac{1}{{D_K}^2}$$
$$= \frac{1}{1024} + \frac{1}{3^2}$$
$$= 0.112$$

This represents a reduction of 8.92 times the number of multiplications. We can also compare the number of trainable parameters in a standard convolution to the number of trainable parameters in a DSC. In a standard convolution, a single kernel has $D_K \times D_K \times M$ parameters. If we have N kernels, we will have a total of $D_K \times D_K \times M \times N$ trainable parameters. In DSC we again look at depthwise convolutions and pointwise convolutions separately. In depthwise convolutions, a single kernel has $D_K \times D_K \times 1$ parameters. Since we need M kernels to achieve full coverage over the entire input volume, we have $D_K \times D_K \times 1 \times M$ total parameters in depthwise convolution. In pointwise convolution, a single kernel has the shape $1 \times 1 \times M$. If we have N kernels in total, we have $1 \times 1 \times M \times N$ total parameters in pointwise convolution. Therefore, the total number of parameters in a DSC is $D_K \times D_K \times 1 \times M + 1 \times 1 \times M \times N$.

Therefore, the reduction in the number of trainable parameters from using DSCs instead of standard convolutions is:

$$\frac{D_K \times D_K \times M \times N}{D_K \times D_K \times 1 \times M + 1 \times 1 \times M \times N}$$
$$= \frac{1}{N} + \frac{1}{{D_K}^2}$$

We can see that the reduction in the number of trainable parameters from using as DSC rather than a standard convolution is the same as that in the number of multiplications.

3.3 Text Detection with SegLink

When provided with a single image, text detection needs to find all text instances within the image and compute their bounding boxes. Although having the same goal as general object detection, text detection is considered a separate computer vision problem. This is because

general object detection algorithms such as YOLO [35], SSD [27], and Faster-RCNN [36] perform poorly on text. The inability of these methods to achieve high accuracies on text detection stems from the orientation and extreme aspect ratios typical of text found in scene images. To combat the extreme aspect ratios and oriented nature of text, many text-specific detection algorithms have taken inspiration from general object detection algorithms, but altered them to perform better. One such method is SegLink [41], the method chosen to detect text in this project.

The architecture of SegLink [41] is based on SSD which put forth the notion of detecting objects on multiple feature layers with convolutional layers (followed by some SoftMax layers). Each feature layer is designed to have a different receptive field so that predictions are made for different sized objects. Specifically, each subsequent feature layer is designed to have half the width and height of the preceding feature layer. However, unlike SSD which directly outputs object bounding boxes, SegLink detects the two types of sub-components of text and combines them together. These two types of sub-components are called segments and links. In simple terms, a segment is an oriented box covering part of a word and a link connects two adjacent segments, denoting that they belong to the same word. Segments and links are detected simultaneously on multiple scales and then combined together afterwards to form words. The original implementation of SegLink used a pretrained VGG16 [45] network for feature extraction. Following SSD, the fully connected layers of VGG16 were converted into convolutional layers, and then additional convolution layers are appended to the network. Inspired by MobileNets [17], we used a similar implementation but replaced some of the standard convolutions with Depthwise Separable Convolutions (DSCs) [44]. This has the effect of reducing the number of trainable parameters from 24M to 7.6M. Figure 3.12 shows the architecture of our implementation.

Figure 3.11: *Example of segments (yellow), and links (green). Image from [41] page 1.*

Type	Stride	Kernel Size	# Kernels	Input Size
Conv	1	3 x 3 x 3	64	512 x 512 x 3
Conv	1	3 x 3 x 64	64	512 x 512 x 64
Max Pool	2	Pool 2 x 2	-	512 x 512 x 64
Conv	1	3 x 3 x 64	128	256 x 256 x 64
Conv	1	3 x 3 x 128	128	256 x 256 x 128
Max Pool	2	Pool 2 x 2	-	256 x 256 x 128
Conv	1	3 x 3 x 128	256	128 x 128 x 128
Conv	1	3 x 3 x 256	256	128 x 128 x 256
Conv	1	3 x 3 x 256	256	128 x 128 x 256
Max Pool	2	Pool 2 x 2	-	128 x 128 x 256
Depthwise Conv	1	3 x 3 dw	256	64 x 64 x 256
Conv	1	1 x 1 x 256	512	64 x 64 x 256
Depthwise Conv	1	3 x 3 dw	512	64 x 64 x 512
Conv	1	1 x 1 x 512	512	64 x 64 x 512
Depthwise Conv	1	3 x 3 dw	512	64 x 64 x 512
Conv	1	1 x 1 x 512	512	64 x 64 x 512
Max Pool	1	Pool 2 x 2	-	64 x 64 x 512
Depthwise Conv	1	3 x 3 dw	512	32 x 32 x 512
Conv	1	1 x 1 x 512	512	32 x 32 x 512
Depthwise Conv	1	3 x 3 dw	512	32 x 32 x 512
Conv	1	1 x 1 x 512	512	32 x 32 x 512
Depthwise Conv	1	3 x 3 dw	512	32 x 32 x 512
Conv	1	1 x 1 x 512	512	32 x 32 x 512
Max Pool	1	Pool 2 x 2	-	32 x 32 x 512
Depthwise Conv	1	3 x 3 dw	512	32 x 32 x 512
Conv	1	1 x 1 x 512	1024	32 x 32 x 512
Depthwise Conv	1	3 x 3 dw	1024	32 x 32 x 1024
Conv	1	1 x 1 x 1024	1024	32 x 32 x 1024
Depthwise Conv	1	3 x 3 dw	1024	32 x 32 x 1024
Conv	1	1 x 1 x 1024	256	32 x 32 x 1024
Depthwise Conv	2	3 x 3 dw	256	32 x 32 x 256
Conv	1	1 x 1 x 256	512	16 x 16 x 256
Conv	1	1 x 1 x 512	128	16 x 16 x 512
Zero Padding	-	-	-	16 x 16 x 128
Conv	2	3 x 3 x 128	256	18 x 18 x 128
Conv	1	1 x 1 x 256	128	8 x 8 x 256
Conv	2	3 x 3 x 128	256	8 x 8 x 128
Conv	1	1 x 1 x 256	128	4 x 4 x 256
Conv	2	3 x 3 x 128	256	4 x 4 x 128
Conv	1	1 x 1 x 256	128	2 x 2 x 256
Conv	2	4 x 4 x 128	256	2 x 2 x 128

Figure 3.12: *Architecture of our modified SegLink. Note, "Depthwise Conv" refers to a Depthwise Convolution, not a DSC, and "dw" refers to a depthwise kernel. Predictions are made just before each pink layer. All layers use "same" padding excpet for the Zero Padding layer and following Conv layer which uses "valid" padding. All layers use the Rectified Linear Unit (ReLu) activation function. Batch Normalization [18] is used after every depthwise convolution and every pointwise convolution.*

3.3.1 Segments

Segments are detected by estimating the confidence scores and geometric offsets to a set of default boxes, as is done in SSD. However in SegLink, only one default box is associated with one feature map location, and its score and offsets are predicted from the features at that location. To generate a segment given a word bounding box and an associated default box (Fig. 3.13a), the word bounding box is first rotated by its angle so that it is axis aligned (Fig. 3.13b). Next, the word bounding box is cropped so that it has the same width as the default box (Fig. 3.13c). Finally, the cropped word bounding box is rotated back by its original angle (Fig. 3.13d). The resulting box is the segment. Given a scene image I with width w_I and height w_I, a location (x, y) on the l-th feature map of size $w_l \times h_l$ corresponds to a default box centered at (x_a, y_a) on the image, where $x_a = w_I(x + 0.5)/w_l$ and $y_a = h_I(y + 0.5)/h_l$. The width and height of the default box is set to a constant size $a_l = 1.5w_I/w_l$.

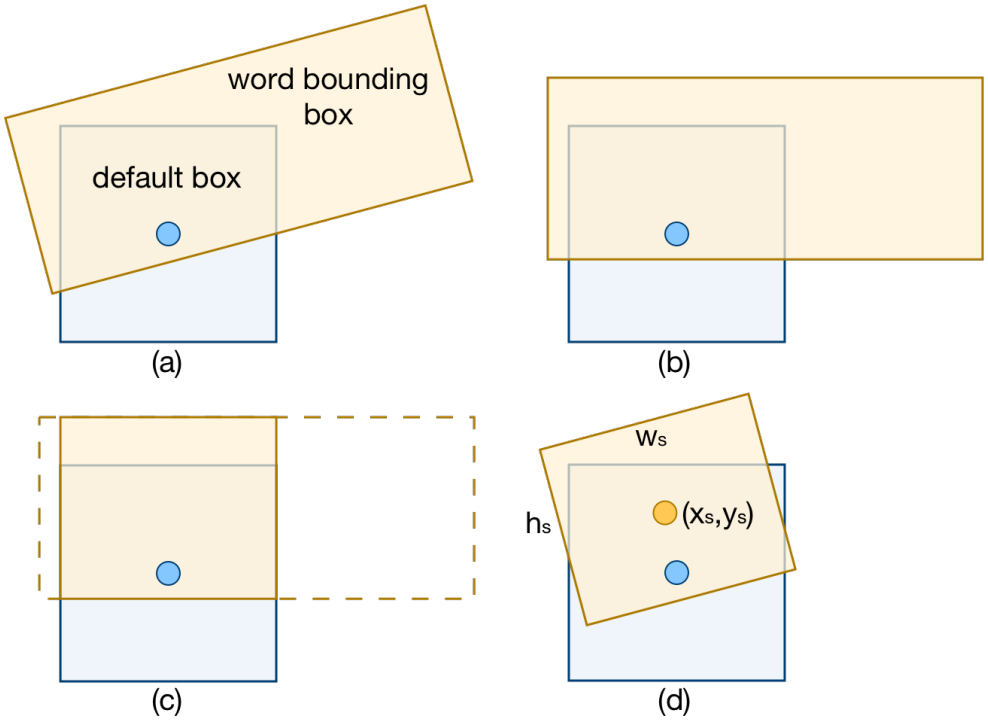

Figure 3.13: *Generating Segments for Training. Image from [41] page 5*

During inference, the convolutional output layers each produce seven channels for segment detection. Two channels represent the segment/non-segment scores, and five chan-

nels represent the geometric offsets from the default boxes. For a given location (x, y) on a feature map, the associated segment $s = (x_s, y_s, w_s, h_s, \theta_s)$ can be calculated using the predicted geometric offsets $(\Delta x_s, \Delta y_s, \Delta w_s, \Delta h_s, \Delta\theta_s)$ by:

$$x_s = a_l \Delta x_s + x_a \tag{3.1}$$

$$y_s = a_l \Delta y_s + y_a \tag{3.2}$$

$$w_s = a_l \times e^{\Delta w_s} \tag{3.3}$$

$$h_s = a_l \times e^{\Delta h_s} \tag{3.4}$$

$$\theta_s = \Delta\theta_s, \tag{3.5}$$

where (x_s, y_s) is the center point of the segment, w_s and h_s is the respective width and height of the segment, and θ_s is the angle of rotation from the horizontal axis. In equations 3.3 and 3.4 the exponential function is used because "Predicting logarithms allows us to output negative values so we can use raw layer activations without an activation function" (B. Shi, personal communication, November 4th, 2018).

3.3.2 Links

SegLink uses two different kinds of links to represent the relationship between segments detected on the same feature layer and on successive feature layers. These are called within-layer links and cross-layer links, respectively. As illustrated in Figure 3.14, for each feature map location, within-layer links are detected for the 8 neighbouring feature map locations. Therefore, each convolutional output layer produces 16 channels, which represent the link/non-link scores for each of the 8 neighbouring locations.

Since SegLink makes detections on multiple layers of different scales, it is possible that the same word may be detected on successive layers. Rather than using Non-Maximum Suppression (NMS) to remove duplicate detections, SegLink uses cross-layer links. Moreover, as illustrated in Figure 3.15, each feature map location has 4 cross-layer neighbours. This property is guaranteed by using either MaxPool, Convolution, or Depthwise Convolution layers with a stride of 2, which halves the size of the subsequent feature map. Therefore, each convolutional output layer produces 8 channels, which represent the link/non-link scores for each of the 4 cross-layer neighbours.

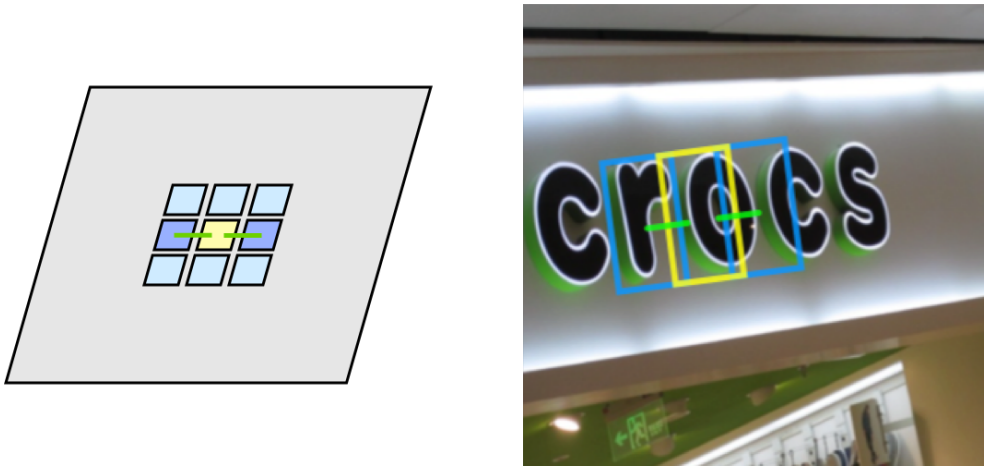

Figure 3.14: *Illustration of within-layer links on feature map (left) and input image (right). Image from [41] page 4*

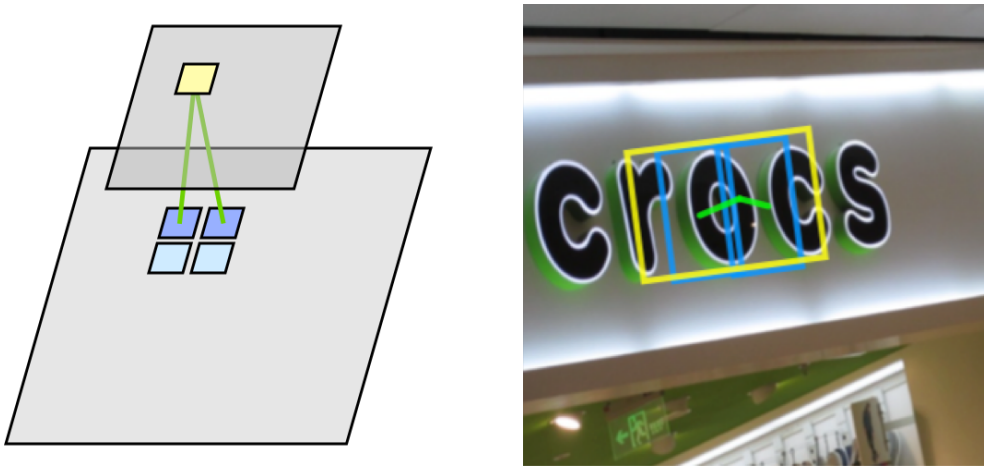

Figure 3.15: *Illustration of cross-layer links on feature map (left) and input image (right). Image from [41] page 4*

3.3.3 The Output

Since SegLink detects segments and links using the same feature maps, the convolutional output layer will be a volume with shape $h_l \times w_l \times 31$ for the feature map layer $l = 1...5$ and shape $h_l \times w_l \times 23$ for the feature map layer $l = 6$. This is because the last output layer will have no subsequent layer to have cross-layer links with.

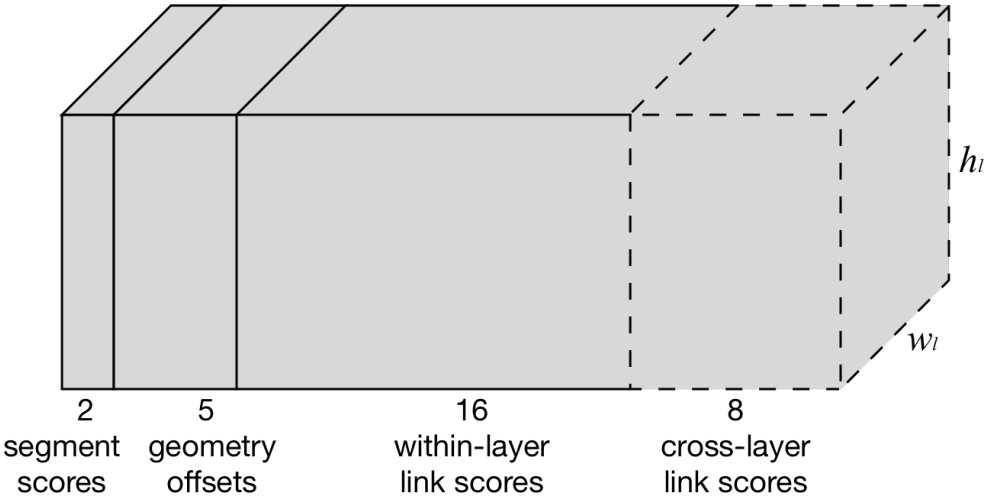

Figure 3.16: *Breakdown of convolutional output. Image from [41] page 4*

3.3.4 Optimization

SegLink is optimized by minimizing the losses of segment classification, offset regression, and link classification. Therefore, a loss function is defined as a weighted sum of all three losses:

$$TotalLoss = L_{segment} + L_{offset} + L_{link} \tag{3.6}$$

$$L_{segment} = \frac{L_{softmax}(y_{segments}, c_{segments})}{N_{segments}} \tag{3.7}$$

$$L_{offset} = \frac{L_{loc}(\hat{s}, s)}{N_{segments}} \tag{3.8}$$

$$L_{link} = \frac{L_{conf}(y_{links}, c_{links})}{N_{links}}, \tag{3.9}$$

where $y_{segments}$ are the labels of all segments, and $y^i_{segments} = 1$ if the i-th default box is labeled as positive, and 0 otherwise. $c_{segments}$ are the predicted segment scores of all segments. The same follows for y_{links}, y^i_{links} and c_{links}. The segment loss $L_{segment}$ and link loss L_{link} are equal to the SoftMax loss $L_{softmax}$ of the predicted segment and

link scores over their groundtruth labels, respectively. Essentially, we want to optimize our confidence in segments being there or not. The geometric offset loss L_{offset} is the Smooth L1 [13] regression loss over the predicted segment geometries $\hat{s}$ and the ground truth segment geometries s. The segment and offset loss are normalized by the number of positive segments $N_{segments}$, and the link loss is normalized by the number of positive links N_{links}.

3.4 Text Recognition with CRNN

The second step in extracting text from a scene image is text recognition. It is the problem of taking a single cropped word image found from text detection and outputting the character sequence of the word. Although there are many ways to perform text recognition, most high accuracy methods follow similar pipelines using Convolutional Neural Networks (CNNs) for feature extraction and Recurrent Neural Networks (RNNs), or sliding windows, to make individual character predictions. One such algorithm which uses this strategy is CRNN [42]. In CRNN, a CNN is first used for feature extraction. The feature maps are then divided into fixed width, vertical slices. These slices are fed into a Bidirectional-Long Short Term Memory (BiLSTM) network to produce individual character predictions. The predicted character sequence is further fed into a Connectionist Temporal Classification (CTC) layer which combines the characters into word predictions.

3.4.1 Convolutional Recurrent Neural Network (CRNN)

The CRNN architecture has three components; feature extraction, sequence labelling, and transcription (Fig. 3.17). For feature extraction, it uses a VGG16-like [45] network, however a small modification is made to the 3rd and 4th max-pooling layers and uses a 1×2 window-size rather than the original 2×2 window-size. This alteration is done in order to both increase the width of feature maps and to help the network distinguish between similar characters such as *"i"* and *"l"*. In this project, we replace some of the standard convolution layers with Depthwise Separable Convolutions (DSCs) [44] to reduce the inference time and model size. This has the effect of reducing the number of trainable parameters from 8.7M to 5.6M. The extracted feature maps are then converted into vertical feature vectors of width one, where the i-th feature vector is a concatenation of the i-th column of the feature maps. The sequence labelling portion of the network is built from two, stacked BiLSTM layers which are used for their ability to use information from distant parts of the image to improve predictions. The input to the BiLSTMs are the vertical feature slices

and the output is the predicted character probabilities for each slice. Finally, the transcription portion of the network takes this output and feeds it into a Connectionist Temporal Classification (CTC) layer to combine the character predications into words. Our modified CRNN architecture is shown in Figure 3.18.

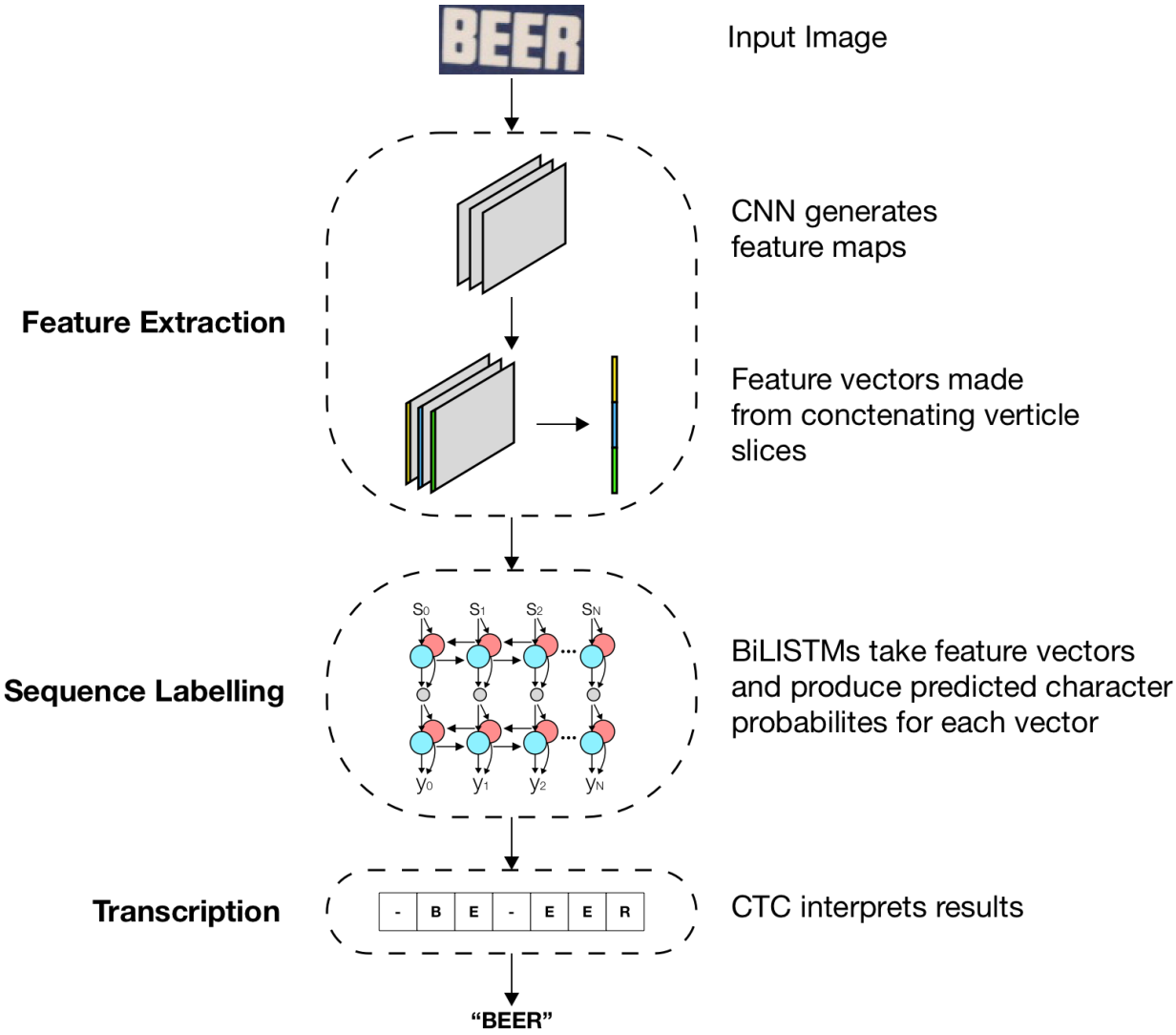

Figure 3.17: *Visualization of CRNN architecture*

Type	Stride	Kernel Size	# Kernels	Input Size
Conv	1	3 x 3 x 1	64	256 x 32 x 1
Max Pool	2	Pool 2 x 2	-	256 x 32 x 64
Conv	1	3 x 3 x 64	128	128 x 16 x 64
Max Pool	2	Pool 2 x 2	-	128 x 16 x 128
Conv	1	3 x 3 x 128	256	64 x 8 x 128
Conv	1	3 x 3 x 256	256	64 x 8 x 256
Max Pool	1 x 2	Pool 2 x 2	-	64 x 8 x 256
Depthwise Conv	1	3 x 3 dw	256	64 x 4 x 256
Conv	1	1 x 1 x 256	512	64 x 4 x 256
Depthwise Conv	1	3 x 3 dw	512	64 x 4 x 512
Conv	1	1 x 1 x 512	512	64 x 4 x 512
Max Pool	1 x 2	Pool 2 x 2	-	64 x 4 x 512
Conv	1	2 x 2 x 512	512	64 x 2 x 512
BiLSTM	-	-	256	64 x 512
BiLSTM	-	-	256	64 x 512
Dense	-	-	87	64 x 512
Softmax	-	Classifier	-	64 x 87

Figure 3.18: *The architecture of our CRNN network. Note, "Depthwise Conv" refers to a Depthwise Convolution, not a DSC, and "dw" refers to a depthwise kernel. All layers use "same" padding excpet for the last Max Pool and Conv layers which use "valid" padding. All layers use the Rectified Linear Unit (ReLu) activation function. Batch Normalization [18] is used after every depthwise convolution and every pointwise convolution.*

3.4.2 Transcription with Connectionist Temporal Classification (CTC)

The sequence labelling portion of the network outputs a matrix in which the element (i, j) represents the probability of predicting the i-th character for the j-th slice (feature vector). Figure 3.19 is an example of what this would look like if we had two slices s_1, s_2 and two possible character predictions a, b. However, we immediately face two problems.

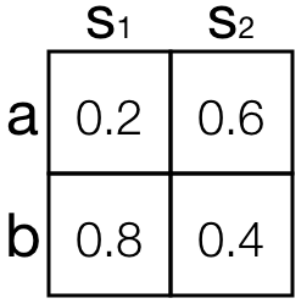

Figure 3.19: *Example of output from BiLSTMs*

Problem 1 - Training the Network

In order to train a network which takes an image as input and outputs a probability matrix as shown in Figure 3.19, the ground truth labels would need to be annotated on an individual feature slice level. For instance, if the input image was Figure 3.20, the ground truth label would need to tell us at exactly which pixel in the horizontal direction the character *"B"* starts and ends. The same follows for each character in the image. This would be very difficult, time consuming, and prone to error.

Figure 3.20: *Example of an input image to CRNN*

Problem 2 - Aligning the Predicted Character Sequence with the Ground Truth Label

Taking the character with the highest probability for each slice as the predicted character, is problematic in aligning the predicted character sequence with the ground truth label

in training. For instance, when considering the image in Figure 3.20, if the predicted character sequence was *"BBBEEEEEERR"* and the ground truth label was *"BEER"*, it would be very difficult to know if the predicted sequence was correct. How would we know that *"BBBEEEEEERR"* maps to *"BEER"* rather than *"BER"*, *"BERR"*, or *"BEEEER"*.

The CTC layer solves both of these problems. Specifically, the CTC layer allows the network to be trained using word labels rather than individual feature slice labels. It also removes the need for post-processing as it aligns the predicted character sequence to the ground truth labels.

How Does it Work?

The CTC layer solves the aforementioned problems using five core ideas. First, it introduces the CTC-blank character, denoted by *"-"*. The CTC-blank character (not to be mistaken for a space in the word) denotes that no character was seen in the current slice. Second, it represents the ground truth label by a set of permutated labels rather than a single label. Third, it creates permutations of the original label by allowing the text to appear at any position. For instance, referring to Figure 3.20, with the original ground truth label *"BEER"*, we would let the permutated ground truth labels be *"BEER--"*, *"-BEER-"*, *"--BEER"*, etc. We do this because we do not know where the text starts and ends. Fourth, it creates permutations by allowing characters to be repeated multiple times. This is done because we do not know how wide each character is, and how many slices it occupies. For example, *"BER---"*, *"BBER--"*, *"-BEERR-"*, etc. Finally, it inserts the CTC-blank character between all label permutations wherever a repeating character originally existed. For example, we get *"BE-ER--"*, *"BEEE-EER"*, etc.

Putting all of these ideas together we are able to generate a set of permutated ground truth labels. To train the network with the CTC layer we calculate the scores for each possible character sequence, then sum over all the scores. This yields the loss for the image/word-label pair.

To decode the predicted character sequence we discard duplicate characters, and then blank characters. For instance, *"--B--EEE-EE-R-"* becomes *"--B--E-E-R-"*, which becomes *"BEER"*.

3.4.3 Optimization

The optimization function of CRNN is:

$$Loss = -\sum_{I_i l_i} \log p(l_i \mid y_i) \tag{3.10}$$

Where I_i is the training image, l_i is the ground truth label, and y_i is the predicted character sequence.

3.5 Text Tracking with Optical Flow

Object tracking is the process of estimating the location of an object in a video frame by using information about the object's location, appearance, and movement from previous frames. Under certain circumstances object tracking can localize objects with similar accuracy to object detection algorithms, while being significantly faster. In the proposed pipeline, optical flow is used as the method for tracking text. Optical flow is the displacement of pixels between video frames, which captures the movement of objects within an image as well as the movement of the camera itself. There are two different kinds of optical flow, namely sparse and dense. Sparse optical flow is where a small region of points are tracked whereas dense optical flow is where points across the entire image are tracked. Obviously, dense optical flow is more computationally intensive yet yields more information.

3.5.1 Bounding Box Translation using Sparse Optical Flow

The Lucas-Kanade (LK) optical flow algorithm [30] provides estimates of the movement of interesting points in successive video frames. Interesting points are defined as points which are unique relative to other nearby points in the image. In this project, we let the interesting points be corners of detected text which are discovered using the Shi-Tomasi corner detection algorithm [43]. Both algorithms use relative pixel intensities to determine the presence of corners and their movement. Once corner points are detected on a previous video frame, the LK algorithm is applied to determine their new locations in the successive frame. Next, we find the median displacement in both the horizontal and vertical directions over all corner points and discard any points with displacements larger than 20% from the median. This is done to remove outlier tracked points. Finally, we calculate the mean displacement of the remaining corner points and shift the bounding box by that amount. Figure 3.21 visually breaks down these steps.

Figure 3.21: *(a) Previous video frame with detected bounding box. (b) Previous video frame with generated corner points. (c) Current video frame with tracked points (yellow) and shifted bounding box (green).*

In addition to the tracking method mentioned above, we also tried tracking text using a variety of features such as Scale-Invariant Feature Transform (SIFT) [29], Maximally Stable Extremal Regions (MSERs) [31], and Histograms of Oriented Gradient (HOG) [11]. Although all of these features produced similar results as optical flow, they took substantially longer time.

3.5.2 Bounding Box Perspective with Optical Flow Grid

Another idea we had was to improve the performance of the tracking algorithm by calculating the change in perspective of the bounding boxes across consecutive video frames. Perspective is how a static object appears from a given distance and angle. Therefore, changing the physical location of the camera will change the appearance of the object. We capture this change in perspective by dividing the detected bounding boxes into multiple regions, observing the magnitude and angle of the predicted optical flow vectors. Specifically, we divided each detected bounding box into a uniform 2×2 grid. For each grid location we generated corner points and calculated the average optical flow vector. Using the relative angle and magnitude of the vectors the shifted bounding box was transformed into a quadrilateral, which more accurately captured the texts new location. Unfortunately, in practice we found this method performed poorly on smaller bounding boxes, as not enough corner points could be generated for each grid location. For this reason, we opted to leave this method out of the final pipeline.

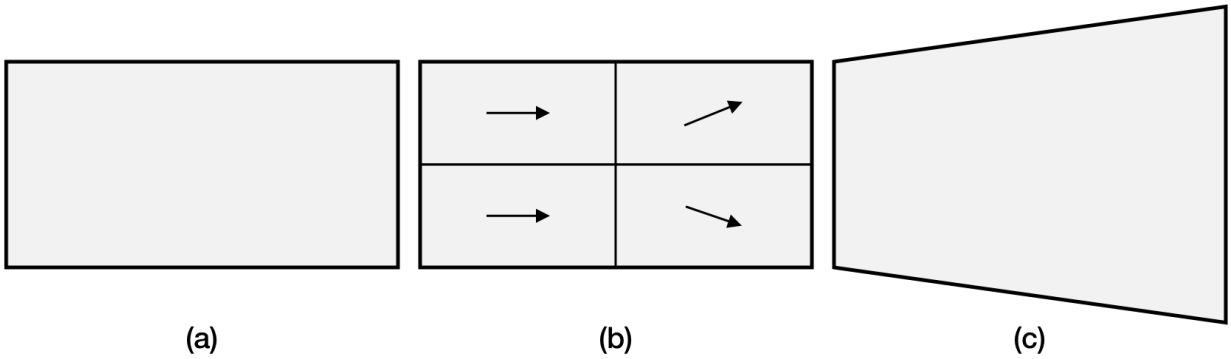

Figure 3.22: *(a) A bounding box. (b) A bounding box is divided into a grid, and the average optical flow vectors are calculated. (c) The perspective of the bounding box is changed.*

3.5.3 Reduction of Detection Search Space using Optical Flow

The final idea we tried was to use optical flow to estimate which areas of the frame remained unchanged from the previous frame, so that these areas could be ignored during the next application of the text detection algorithm. Ideally, this would lead to reduced detection times and thus reduced text extraction times. To do this we resized the previous and current frames to 150×150 pixels and divided them into a 3×3 grid. Next, we generated corner points for each grid location and tracked them as was done in the aforementioned tracking methods. Finally, using a set of rules we could infer which regions of the current frame had already been detected and did not need to be re-detected. Figure 3.23 visually depicts the method for determining which area of the image does not need to be detected.

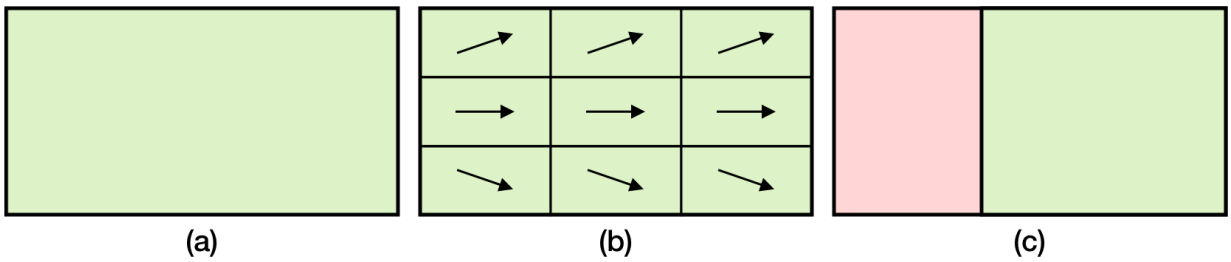

Figure 3.23: *(a) Previous frame. (b) Previous frame with optical flow vectors for each grid location. (c) Current video (red), and portion of the previous frame (green) which has not changed (only the red part needs to be detected, the green part can be tracked).*

However, this solution is only applicable in a small number of situations. It only works when the camera movement follows a nearly perfect horizontal translation, vertical

translation, or inwards zoom. In Figure 3.24a, the changed portion of the current frame is depicted in red and the unchanged portion is depicted in green, after a diagonal camera movement. Cropping out this unchanged region leaves a non-rectangular image which cannot be given as input to the text detection algorithm. Blacking or whiting out the unchanged region yields a rectangular image, but of the same initial dimensions, which results in the same detection times. Another idea shown in Figure 3.24b and Figure 3.24c, is to split the changed region into smaller rectangular regions, perform detections on those, and then combine their results. However, text appearing in multiple sub-regions are often not detected or are detected as word fragments, leading to further post- processing. Altogether, these drawbacks make this method ideal for a stationary cameras or cameras on robots, but not for handheld video. Since the analysis section deals with handheld video, this method is omitted from the pipeline.

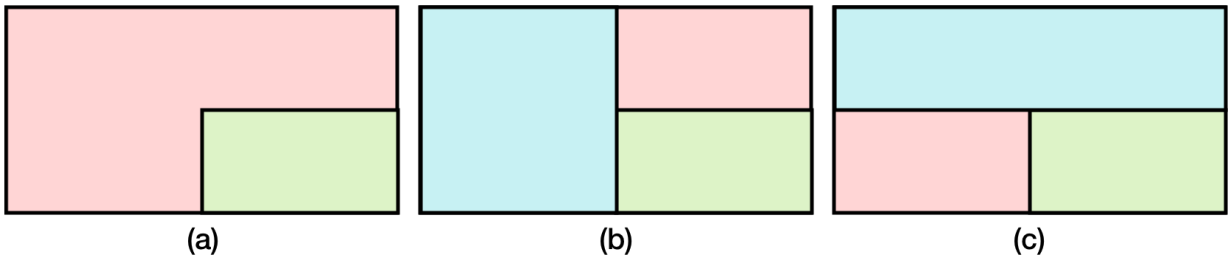

Figure 3.24: *(a) Changed current frame (red), and unchanged current frame (green). (b,c) Changed current frame (red), unchanged current frame (green), changed sub region of current frame (blue).*

3.6 Natural Language Model

Ideally, the text detection algorithm would predict the pixel-perfect bounding boxes for all text instances and the text recognition algorithm would correctly recognize the corresponding character sequences. However, in practice this is often not the case. Typically, the algorithms extract mostly correct character sequences with a few misplaced, missed, or incorrect characters. To mitigate this problem, we introduce a probabilistic language model which takes potentially incorrect character sequences and returns the most likely correct character sequence.

Before designing a language model, we need to understand what kinds of imperfect character sequences are likely to occur. When examining the detection results of SegLink [41], we notice that two closely located words will often be detected as one large word.

Examples of this can be seen in Figure 3.25a and Figure 3.25d. This is due to the linking threshold being set to a non-optimal value for that particular image, which is one of the primary drawbacks of the algorithm. Another common issue is that the detected bounding box will cut-off part of the word or will include the end of a neighbouring word, as seen in Figure 3.25b. The third common issue is when the text recognition algorithm simply misclassifies the detected character sequence, shown in Figure 3.25c.

Figure 3.25: *Example output of SegLink (red bounding boxes) and CRNN (yellow character sequences) on static images*

Our initial idea to improve recognition results was to use a bag-of-words and/or spell-checking model. However, we quickly realized that this alone would not yield optimal results, due to the detection algorithm mistakenly identifying multiple words as one. For example, spell-checking the text *"millantiques"* from Figure 3.25d would either result in *"atlantiques"* (a region in southwest France), or simply the original string *"millantiques"* if no good match was found. Therefore, we decided to first apply a text segmentation algorithm to the predicted character sequence before applying a spell-checking model.

3.6.1 Text Segmentation

Given the string *"treewalkpark"* the ideal segmentation would yield *"tree"*, *"walk"*, *"park"*. Our solution to this problem was to use the Naïve Bayes rule. This rule simply states that the relative probability that a candidate segmentation is the correct segmentation is equal to the product of the probabilities of each segmented text being a real word. A candidate segmentation is one possible way to split a string into two or more substrings. For example, the relative probability of the candidate segmentation *"tree""*, *"walk"*, *"park"* being the correct segmentation of *"treewalkpark"*, is equal to the probability that *"tree"* is a real word, times the probability of *"walk"* being a real word, times the probability that

"park" is a real word. To determine how likely a string is to be a real word, we use a word-frequency list which tells us how frequently each word appears in a corpus of text. In our project, we use a word-frequency list [34], which lists the 333,333 most common words in the English language and their frequencies, in a corpus of one trillion words gathered from the internet. For our purposes, this works well as the list covers about 98% of all words, including common slang terms and acronyms, and takes 5Mb of disk space.

The next hurdle was deciding how to generate the candidate segmentations. The naïve approach would be to generate all possible segmentations, however, this is infeasible since there are 2^{n-1} possible segmentations for a string of length n. Another idea was to take the same approach a human would take when segmenting text, which involves performing a scan of the text from left to right, asking *"does the first character(s) of the text sound like a word I know"* until eventually they say *"yes"*. At this point, a human would look at the remainder of the text and perform another scan. This process repeats until the entire string has been traversed. An example of the thought process of a human when segmenting the string *"treewalkpark"* is shown in Table 3.1.

Table 3.1: Example thought process of a human when attempting to segment the string *"treewalkpark"*

Question: Is this a word I know?	Answer
treewalkpark	no
t	no
tr	no
tre	no
tree	**yes**
walkpark	no
w	no
wa	no
wal	no
walk	**yes**
park	**yes**

In practice the text segmentation algorithms implementation is slightly different than the method shown in Table 3.1, but the core idea remains similar. Specifically, the implementation compares the relative probability of the original string *"treewalkpark"* to the relative probability of the $n-1$ segmentations with one split. Of these, the segmentation with the highest relative probability is chosen, and then we recursively reinitialize this

process, using the two substrings as the original string.

3.6.2 Spell Correction

Once the character sequence is divided into its compromising words, the next task is to ensure that these new words are valid English words. To do this, we run a spell correction algorithm on each of the new words. The spell correction algorithm creates a set of words with edit distances of one and two from the original text, that appear in a predefined lexicon. The edit distance is the number of changes that are made to one string in order to transform it into a different string. For example, the edit distance between *"pizza"* and *"pozza"* is one, and the edit distance between *"pizza"* and *"pozzo"* is two. From this set, the word with the highest probability is selected as the correct word, where the probability is defined the same as in the text segmentation algorithm.

3.6.3 Utilizing Previous Guesses

To further enhance the recognition accuracy of the pipeline, we incorporate previously recognized character sequences to enhance current predictions. Specifically, in the spell correction stage, if one of the current candidate corrections matches the previously output word, then its probability is increased for the current prediction. For example, if the recognized character sequence for the current frame is *"happer"*, possible spell corrections will include *"happen"*, and *"happier"*. Between these two words, *"happen"* has the higher probability, and would normally be chosen. However, if the previous frames prediction was happier then the likelihood of *"happier"* being the optimal correction in the current frame will be increased, and subsequently chosen as the spell correction.

Chapter 4

Performance and Evaluation

In this section we explore the datasets used for training and evaluation, discuss implementation details of the SegLink [41] and CRNN [42] algorithms, and evaluate the performance of each pipeline component individually and as a whole. Moreover, we investigate the weaknesses of pipeline components and discuss potential improvements.

4.1 Datasets

SynthText in the Wild (SynthText) [15] is the largest scene text dataset with more than 800,000 synthetic images. Images are created by merging real scene images with artificially generated text. Moreover, quadrilateral word, character and text-line bounding boxes, and character sequences are provided for each scene image. We used this dataset for training SegLink with whole images, and for training CRNN by cropping each word from the images.

Home Video (HV) is a single video containing 644 frames of text signage including street signs, shopping bags, street addresses, and license plates. The video was captured by an iPhone 6 in 1080p at 30fps and has been annotated on a word-level for all frames. The video is used in this project to test the performance of the individual pipeline components as well as the overall pipeline.

4.2 Training and Implementation

One of the most challenging aspects of this project was training the neural networks due to the depth of the network architectures and the size of the datasets used. Subsequently, we needed a powerful machine to train the networks in a timely manner. After experimenting with other cloud-based solutions (Google Collab), we chose to use Compute Canada [7] for all network training. Compute Canada is a nationwide collection of Advanced Research Computing (ARC) systems available to the Canadian academic community for research purposes. Compute Canada provides access to multiple machines with Intel E5-2683 V4 CPUs (2.1Ghz), Nvidia Tesla P100 Graphics Cards, and 128Gb of RAM. Additionally, to implement our pipeline we used Python v3.6.4, Keras v2.1.3 [9] (with TensorFlow [1] backend), and OpenCV v3.3.1 [4], as well as code [39, 34, 32, 19] provided by several authors.

4.2.1 SegLink Implementation Details

The SegLink text detection algorithm was trained on the SynthText dataset and was optimized by the Stochastic Gradient Descent (SGD) algorithm with a momentum of 0.9. The first 8 layers of the model were initialized with pre-trained VGG16 [45] weights and then frozen, while the remaining layers were randomly initialized. Images were resized to 512×512 after a random cropping as was done in SSD [27]. The learning rate was set to 10^{-3} and a batch size of 16 was used. For evaluation, the segment and link thresholds are set to the optimal values which are found by performing a grid search with step size of 0.1. For both datasets, the optimal segment and link threshold are 0.6 and 0.3, respectively.

4.2.2 Depthwise Separable Convolution SegLink (DSC SegLink) Implementation Details

The DSC SegLink algorithm was trained on the SynthText dataset. Due to the lack of initial weights for the Depthwise Separable Convolution (DSC) layers, training the DSC SegLink model proved to be challenging. While the unchanged layers of the network were initialized with weights from the SegLink model trained on SynthText, the DSC layers were randomly initialized. The network was trained multiple times using various optimization algorithms, learning rates, momentums, and decays. However, as shown in Table 4.1, under no configuration did the algorithm converge to comparable performance as the baseline.

4.2.3 CRNN Implementation Details

The CRNN text recognition algorithm was trained on the SynthText dataset in which input images were found by cropping the word bounding boxes from the scene images, then resized to $256 \times 32 \times 1$. The convolution layers were initialized with pre-trained VGG16 weights, while the other layers were initialized randomly. It was optimized by the SGD algorithm with a momentum of 0.9. The learning rate was set to 10^{-2} and a batch size of 128 was used.

4.2.4 Depthwise Separable Convolution CRNN (DSC CRNN) Implementation Details

The DSC CRNN algorithm was trained on the SynthText dataset in the same way as the CRNN algorithm. The DSC layers were randomly initialized while all other layers were initialized with CRNN weights trained on the SynthText dataset, then frozen. All other training parameters remained the same as before.

4.3 Evaulation

4.3.1 Evaluation Metrics

The metrics we used to evaluate both the components of our pipeline, individually and combined, were Precision (Eqn. 4.1), Recall (Eqn. 4.2), F-Measure (Eqn. 4.3), and Accuracy (Eqn. 4.4).

$$P = \frac{TP}{TP + FP} \tag{4.1}$$

$$R = \frac{TP}{TP + FN} \tag{4.2}$$

$$F = 2 \times \frac{P \times R}{P + R} \tag{4.3}$$

$$A = \frac{TP + TN}{TP + TN + FP + FN}, \tag{4.4}$$

where *TP*, *TN*, *FP*, *FN* are the number of True Positives, True Negatives, False Positives, and False Negatives respectively. For text detection, *TP*, *FP* and *FN* are the number of correctly detected boxes, incorrectly identified boxes, and missed boxes respectively. A predicted box is considered correctly detected if it has an Intersection over Union (IoU) ratio larger than 0.5. For text recognition and end-to-end text extraction, *TP* is the number of correctly predicted correct words; *TN* is the number of correctly predicted incorrect words; *FP* is the number of incorrectly predicted correct words; and *FN* is the number of incorrectly predicted incorrect words.

4.4 Testing Hardware

All tests were carried out on a 2012 iMac with a 2.9 GHz 3rd Generation Intel Core i5 processor and 16 Gb of DDR3 RAM. Testing was done on this machine rather than on Compute Canada to more accurately reflect the frame rate which may be achievable by a mobile device.

4.5 Results

The comparison of the original SegLink model against the DSC SegLink model is shown in Table 4.1. Due to the lack of pretrained DSC layer weights, the DSC SegLink model failed to converge. However, should these weights become available, the DSC SegLink model is considerably faster and smaller than the original SegLink model.

Table 4.1: Comparison of the original SegLink model with the modified DSC SegLink model

	SynthText				
Method	P	R	F	FPS	Size (Mb)
SegLink	**0.889**	**0.788**	**0.835**	0.3912	97.6
DSC SegLink	0.0000	0.0000	0.0000	**0.4931**	**30.9**

Table 4.2 shows the performance of the original CRNN and DSC CRNN on a subset of the SynthText dataset. The DSC CRNN algorithm shows a 36% reduction in inference

time and 2% drop in accuracy over the original model. This result both verifies the implementation of the DSC and the hypothesis that DSCs can be used for the problem of text extraction.

Table 4.2: Comparison of the original CRNN model with the modified DSC CRNN model

	SynthText		
Method	Accuracy	FPS	Size (Mb)
CRNN	**0.9570**	1.80	35.0
DSC CRNN	0.9393	**2.45**	**22.5**

The results of our entire pipeline are shown in Table 4.3. The first entry in the table illustrates the performance of the SegLink and CRNN algorithm when run on every frame of the Home Video (HV). Surprisingly, the addition of the language model to the pipeline algorithm reduced the text extraction accuracy. After inspection, we determined this was due to the misrepresentation of the kinds of text found in scene images. Specifically, in the home video, and in most scene images, text signage typically consists of business names, people names, and acronyms such as *Starbucks*, *McMaster*, and *Hamilton*. However, the dictionary used by the language model contained only the most common English words such as *coffee*, *store*, *school*, and *city*. This misrepresentation of what typical words are utilized, caused the language model to incorrectly modify extracted text. In the next line of Table 4.3 we applied the same language model but added the words we found in the video to the language models dictionary (LM*). When this dictionary was used, text extraction accuracy was much higher, illustrating the importance of a robust dictionary. Next, we applied our tracking algorithm. This has the effect of drastically reducing the inference time while experiencing a small drop in accuracy. Combining the modified language model (LM*) with the tracking algorithm, the pipeline is capable of being faster and more accurate than the SegLink and CRNN baseline. Finally, we demonstrate the effect of the DSC models. By using the DSC CRNN algorihtm, the pipeline is able to run faster while taking less memory, however as mentioned previously, the DSC SegLink model is unusable. Overall, the pipeline using the original SegLink algorithm, the DSC CRNN algorithm, optical flow tracking, and the language model with the robust dictionary, improved on the baseline accuracy, speed, and model size by 10.6%, 212.5%, and 4.8%, respectively.

Table 4.3: Comparison of the end-to-end video text extraction performance of the original SegLink+CRNN model our modified SegLink+CRNN model, and our entire pipeline

	VIDEO				
Method	P	R	F	FPS	Size (Mb)
SegLink+CRNN (baseline)	0.3476	0.3428	0.3451	0.3774	132.6
SegLink+CRNN+LM	0.2414	0.3065	0.2701	0.3748	138.9
SegLink+CRNN+LM*	**0.4202**	**0.4143**	**0.4173**	0.3767	139.0
SegLink+CRNN+Tracking	0.3332	0.3131	0.3224	0.8111	132.6
SegLink+CRNN+LM*+Tracking	0.3981	0.3752	0.3863	0.8056	139.0
SegLink+DSC CRNN+LM*+Tracking	0.3934	0.3708	0.3817	**0.8228**	126.5
DSC SegLink+DSC CRNN	0.0000	0.0000	0.0000	0.4914	**53.5**

Chapter 5

Conclusions and Future Work

In this report, we presented a first step in building a fast and accurate text extraction pipeline. The pipeline using the original SegLink algorithm, the DSC CRNN algorithm, optical flow tracking, and the language model with the robust dictionary, improved on the baseline accuracy, speed, and model size by 10.6%, 212.5%, and 4.8%, respectively. Future work should apply the pipeline to mobile devices and make improvements to text algorithms, training processes, tracking algorithms, natural language model, and overall pipeline speed.

5.1 Text Extraction Algorithms

Despite the high accuracy and fast inference times of the SegLink [41] text detection algorithm, there are some drawbacks. First, both the segment and link thresholds need to be set manually. Second, SegLink is incapable of detecting text with large character spacing or curved orientation. Finally, SegLink requires a large amount of post processing, namely a depth-first search over all segments and links, a Non-Maximum Suppression (NMS), and a geometric combination of segments and links into word-level bounding boxes.

At the onset of this project the literature survey suggested that the speed of SegLink was superior to TextBoxes++ [25], which we now believe to be untrue. TextBoxes++ has a similar SSD-based [27] architecture, performance, and inference time as SegLink, but rather than predicting components of words and combining them together, TextBoxes++ directly outputs word-level bounding boxes. Therefore, this algorithm addresses and solves all of the drawbacks of SegLink. Moving forward, we believe that TextBoxes++ would be

a better choice for text detection as it would be substantially easier to implement on a mobile device. Unfortunately, for this project we did not choose TextBoxes++ for the detection algorithm because we perceived it as being slower than SegLink. Additionally, as a possible improvement to text recognition, modifying the CRNN [42] algorithm with the addition of a Spatial Transformer Network (STN) would allow CRNN to rectify slanted or curved text before recognizing it [48].

Segmentation-based text extraction algorithms such as Fast Oriented Text Spotting (FOTS) [28] show promising future opportunities. FOTS jointly detects and recognizes text from scene images using the same feature maps. This has several advantages over the aforementioned methods, namely it allows for the application of feature map-level tracking such as Feature-guided Flow Aggregation (FGFA) [52] and Deep Feature Flow (DFF) [53] to both the detection and recognition feature maps. This would produce a drastic increase in overall text extraction accuracy in a video context. Additionally, FOTS is capable of detecting and recognizing curved text unlike TextBoxes++ or SegLink. We did not initially consider FOTS for the pipeline because the lack of available documentation and our understanding of the previous research. Moving forward, we strongly recommend using a method such as FOTS which uses the same features for both text detection and recognition.

5.2 Tracking

In our project, we utilized optical flow to reduce the total computation of the pipeline in order to reduce computation, however, tracking can also be used to improve accuracy. FGFA provides a solution to align and combine previous frame feature-maps with current frame feature maps in order to increase accuracy on frames with blurred text.

5.3 Natural Language Model

For this project, we used a probabilistic language model to segment and correct recognized character sequences, which has the effect of increasing the recognition accuracy on individual word predictions. However, the language model does not consider other recognized text within the frame to improve predictions. Additionally, the model does not take any visual cues from the image such as the environment in which the text appears. Future research should address these concerns.

5.4 Faster Network Design

Our pipeline utilizes DSCs [44] to both reduce inference time and model size. A recent paper [40] demonstrated the ability of networks with DSCs to be quantized, allowing for further model size reduction. Huffman coding and pruning can also be used to reduce the models size [16]. Furthermore, MobileNetsV2 [46] introduces improvements to DSC networks which are applicable to the models.

5.5 Mobile App

A fast and accurate text extraction mobile application has uses in text-based geolocation, multi-language translation, crowd-sourcing, and robotics. Integrating our pipeline into a multi-platform application can be done natively using TensorFlow [1] and Keras [9], and is an exciting next step.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems. https://www.tensorflow.org/, 2015.

[2] C. Bartz, H. Yang, and C. Meinel. Stn-ocr: A single neural network for text detection and text recognition. *CoRR*, abs/1801.02765, 2017.

[3] G. Bertasius, L. Torresani, and J. Shi. Object detection with spatiotemporal sampling networks. *CoRR*, abs/1803.05549, 2018.

[4] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[5] M. Busta, L. Neumann, and J. Matas. Deep textspotter: An end-to-end trainable scene text localization and recognition framework. *ICCV*, 2017.

[6] S. Liu C. Hetang, H. Qin and J. Yian. Impression network for video object detection. *CoRR*, abs/1712.05896, 2017.

[7] Compute Canada. https://www.computecanada.ca, 2016.

[8] X. Chen, Z. Wu, and J. Yu. Temporal single-shot object detection based attention-aware lstm. *CoRR*, abs/1803.00197, 2018.

[9] François Chollet et al. Keras. https://github.com/fchollet/keras, 2015.

[10] J. Dai, H.Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei. Deformable convolutional networks. *ICCV*, pages 764–773, 2017.

[11] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. *CVPR*, pages 886–893, 2005.

[12] P. Fischer, A. Dosovitskiy, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. Flownet: learning optical flow with convolutional networks. *CoRR*, abs/1504.06852, 2015.

[13] R. B. Girshick. Fast r-cnn. *ICCV*, 2015.

[14] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CVPR*, 2014.

[15] A. Gupta, A. Vedaldi, and A. Zisserman. Synthetic data for text localization in natural images. *CVPR*, 2016.

[16] S. Han, H. Mao, and W. J. Dally. Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149, 2015.

[17] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyandand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[18] S. Ioffe and C. Szegedy. Batch normalization: accelerating deep network training by reducing internal covariate shift. *arXiv Preprint*, 2015.

[19] j2kun. Text segmentation. https://github.com/j2kun/segment/blob/master/segment.py, 2013.

[20] Y. Jiang, X. Zhu, X. Wang, S. Yang, W. Li, H. Wang, P. Fu, and Z. Luo. R2cnn: rotational region cnn for orientation robust scene text detection. *CoRR*, abs/1706.09579, 2017.

[21] V. Kalogeiton, P. Weinzaepfel, V. Ferrari, and C. Schmid. Action tubelet detector for spatio-temporal action localization. *ICCV*, pages 4415–4423, 2017.

[22] D. Karatzas, L. Gomez-Bigorda, A. Nicolaou, S. K. Ghosh, A. D. 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. *ICDAR*, pages 1156–1160, 2015.

[23] D. Karatzas, F. Shafait, S. Uchida, M. Iwamura, L. G. Bigorda, S. R. Mestre, J. Mas, D. F. Mota, J. A. Almazan, and L. P. de las Heras. Icdar 2013 robust reading competition. *ICDAR*, pages 1484–1493, 2013.

[24] Y. Li, W. Jia, C. Shen, and A. van den Hengel. Characterness: An indicator of text in the wild. *IEEE Trans. Image Processing*, 23(4):1666–1677, 2014.

[25] M. Liao, B. Shi, and X. Bai. Textboxes++: A single-shot oriented scene text detector. *CoRR*, abs/1801.02765, 2018.

[26] M. Liu and M. Zhu. Mobile video object detection with temporally-aware feature maps. *CoRR*, abs/1711.06368, 2017.

[27] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, and S. E. Reed. Ssd: single shot multibox detector. *ECCV*, 2016.

[28] X. Liu, D. Liang, S. Yan, D. Chen, Y. Qiao, and J.Yan. Fots: Fast oriented text spotting with a unified network. *CoRR*, abs/1801.01671, 2018.

[29] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Compute Vision*, 6(2), 2004.

[30] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. *DARPA*, pages 121–130, 1981.

[31] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image of Vision Computing*, 22(10):761–767, 2004.

[32] mvoelk. Ssd detectors. https://github.com/mvoelk/ssd_detectors, 2016.

[33] G. Ning, Z. Zhang, C. Huang, X. Ren, H. Wang, C. Cai, and Z. He. Spatially supervised recurrent convolutional neural networks for visual object tracking. *ISCAS*, pages 1–4, 2017.

[34] P. Norvig. http://norvig.com.

[35] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *CVPR*, 2016.

[36] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *NIPS*, 2015.

[37] X. Rong, C. Yi, X. Yang, and Y. Tian. Scene text recognition in multiple frames based on text tracking. *ICME*, pages 1–6, 2014.

[38] J. Sanchez, F. Perronnin, T. Mensink, and J. Verbeek. Image classification with the fisher vector: Theory and practice. *International journal of computer vision*, 105(3):222–245, 2013.

[39] sbillburg. Crnn with stn. https://github.com/sbillburg/CRNN-with-STN, 2018.

[40] T. Sheng, C. Feng, S. Zhou, X. Zhang, L. Shen, and M. Aleksic. A quantization-friendly separable convolution for mobilenets. *CoRR*, abs/1803.08607:761–767, 2018.

[41] B. Shi, X. Bai, and S. J. Belongie. Detecting oriented text in natural images by linking segments. *CVPR*, pages 3482–3490, 2017.

[42] B. Shi, X. Bai, and C. Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text. *IEEE TPAMI*, 39(11):2298–2304, 2017.

[43] J. Shi and C. Tomasi. Good features to track. *CVPR*, 1994.

[44] L. Sifre. Rigid-motion scattering for image classification. *PhD thesis*, 2014.

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

[46] M. Snadler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Mobilenet v2: Inverted residuals and linear bottlenecks. *CoRR*, abs/1801.04381, 2018.

[47] Z. Tian, W. Huang, T. He, P. He, and Y. Qiao. Detecting text in natural image with connectionist text proposal network. *ECCV*, 2016.

[48] F. Yin, Y. Wu, X. Zhang, and C. Liu. Robust scene text recognition with automatic rectification. *CVPR*, pages 4168–4176, 2016.

[49] F. Yin, Y. Wu, X. Zhang, and C. Liu. Scene text recognition with sliding convolution character models. *CoRR*, abs/1709.01727, 2017.

[50] X. Zhou, C. Yao, H. Wen, Y. Wang, S. Zhou, W. He, and J. Liang. East: an efficient and accurate scene text detector. *CVPR*, pages 2642–2651, 2017.

[51] X. Zhu, J. Dai, X. Zhu, Y. Wei, and L. Yuan. Towards high performance video object detection for mobiles. *CoRR*, abs/1804.05830, 2018.

[52] X. Zhu, Y. Wang, J. Dai, L. Yuan, and Y. Wei. Flow-guided feature aggregation for video object detection. *ICCV*, pages 408–417, 2017.

[53] X. Zhu, Y. Xiong, J. Dai, and Y. Wei. Deep feature flow for video recognition. *CVPR*, pages 4141–4150, 2017.