Text Snake and Thin Plate Spline Transformation for Text Recognition Model

Anonymous Authors[1]

Abstract

Detecting and recognizing text in a natural setting is a challenging task due to various shapes and unpredictable environment these texts are a part of. Most methods have successfully detected text instances in an environment despite the environment's complexity. However, most methods assume the text instances are somewhat straight or in a linear format. However, it is quite common to find text that is "curved" or non-linear in the real world. Therefore, applying a text recognition model on the output of a text detection model using STN fails to achieve desired results. Furthermore, even after detecting the text, one must rectify the text before feeding it into the text recognition model. Therefore, I propose a method that combines a previous text detection model((Long et al., 2018)) with a text rectification algorithm: Thin Plate Spline (TPS) and a text recognition model((Bartz et al., 2018)) to identify curved text in natural settings.

1. Brief Summary

Three previous state of the art methods that have successfully detected and recognized text include: SEE: Towards Semi-Supervised End-to-End Text (Bartz et al., 2018), Symmetry-Constrained Rectification Network for Scene Text Recognition (Yang et al., 2019), and Mask TextSpotter: An End-to-End Trainable Neural Network for Spotting Text with Arbitrary Shapes (Lyu et al., 2018). At a high level, the relationship between each of the papers is the problem they are trying to solve. The problem is too detect and recognize text in the "wild". This means finding images from natural settings such as street and traffic signs, billboards, etc. At a deeper level, the relationship between these papers shows how they tackle this problem. There are two components to working with finding text in natural settings: text detection and text recognition. Different papers state that they should tackle the problem by either working separately on text detection and text recognition, while other papers say to treat these two processes as one problem.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

2. Related Work

2.1. SEE: Towards Semi-Supervised End-to-End Text

2.1.1. Summary

According to (Bartz et al., 2018), the authors suggest a single deep neural network "that learns to detect and recognize text from natural images, in a semi-supervised way". In essence they have created one network to tackle the problem of both recognizing and detecting text in a neural network called SEE. A neural network can learn from another neural network, and the authors of this paper have taken that to advantage. SEE is a Deep Neural Network that is comprised of two networks that integrates and jointly learns from them. The first network is a spatial transformer network (STN) and the second is a text recognition network (TRN). A spatial transform network's purpose is to change an image using spatial transformations (such as Affine, Projective, TPS etc.), to enhance geometric in-variances in the image (Tsang, 2019). SEE uses STN to learn how to detect regions of an image as in real life, the text is not always shown straightforward. Sometimes it is warped, circular, or taken at an angle that makes it hard to read. After the STN, the TRN can be used to now detect the text in the image as the STN has corrected the image to the TRN's advantage.

2.1.2. Critique

However, there are some drawbacks to this network. The architecture of their network is very simple, but actually training it requires certain data sets. One cannot just take random pictures and have the network train on it. The authors stated that they had to choose certain images and feed this carefully chosen images in the network. Otherwise it does not work as intended. However, if done correctly it gives competitive results. The authors compare their method to other methods accuracy of the SVHN data set and the FSNS data set. They show that their results compared to two state of the art networks.

While their results are not the highest, their whole

purpose is to show that even if they combine the text recognition and text identification networks into one DNN it can still provide promising results similar to those who treat identification and recognition as two separate tasks. Their method is still computationally more efficient and still accurate as well. However, another fault of this architecture is that it does not work as well in the FSNS data set. Although the accuracy was somewhat in the middle between the two state of the art methods, its drawback is that the model is forced to recognize a limited number of works. Therefore, if there is more text in the image, the model ignores it. Some questions I have for this article is exactly what is their criteria for an image that would be good to train this model. Is there a certain way we can calculate what image is "good" and what image is "bad"? Another critique I would like to add is that this article did not test their architecture on a lot of data sets. There are many data sets that are available to the public to test their model (IIIT5K, SVT, IC03, IC13, IC15, SVTP, CUTE etc.) However, the authors in this paper tested their method on 2-3 data sets. Therefore, I am not too sure about the robustness of their architecture.

2.2. Symmetry-Constrained Rectification Network for Scene Text Recognition

2.2.1. Summary

However, according to (Lyu et al., 2018) tackling text detection and recognition separately might be better and actually leads to better results. (Lyu et al., 2018) proposes adding a text rectification module by using a Symmetry-constrained Rectification Network (ScRN) than applying a recognition module to find the word from the image. They claim ScRN performs very well and still does not require extra computation. Diving deeper into the method, there are three main components to (Lyu et al., 2018) method: the backbone, the rectification module, and the recognition module. The backbone of the architecture is FPN (Feature Pyramid Network) coupled with ResNet-50. This is used by both the rectification and the recognition module. The rectification module produces dense pixel-wise predictions of geometric attributes. This is how feature maps are "rectified" as regular ones via TPS (a tool for modeling coordinate transformation). The geometric attributes of a text is the scale orientation, character orientation, and text orientation. These three attributes are extracted and then inputs into a TPS (Thin-Plate-Spline). This transformation takes the points given from the geometric attributes, and straightens the crooked or out of shape words in the image. Now an enriched feature map is inputs into the recognition module. The recognition module is comprised of 4 convolution layers, where the layers further encode the image and reserve more discernible features of the text, and a bidirectional LSTM. All together, this architecture is called a Symmetry-constrained Rectification Network (an ScRN).

2.2.2. Critique

Critiquing the paper there are both cons and pros to it. The cons of this architecture is that the "rectification module suffers from the curved text whose terminal characters have a nearly horizontal orientation and are close to the image borders," (Lyu et al., 2018). This means that if the image has text that spills over the boards or have words where the middle letters are curved and the end letters are horizontal, the capture word misses those horizontal oriented letters. However, the pros of this paper is that according to the 7 data sets: IIIT5K, SVT, IC03, IC13, IC15, SVTP, CUTE, comparing it to existing methods, their method scored higher or was second by 0.06 at most. This shows that their architecture performed very well and was also had less computation than the existing state of the art methods.

My questions for this article primarily concerned with the computation time. This claims that it is computationally better but does not give any data to support this. And since one of the future project concerning this article is combining it with an end-to-end system that can deal with text-recognition (the SEE project), they should address the computation time as I would like to accomplish this task. My other question is why does this not work words that are in a straight line? It seems that that the images they had most trouble with were words that were considered straight. And if this is the case, why not consider these as two separate tasks and deal with them accordingly?

2.3. Summary and Critique of Mask TextSpotter: An End-to-End Trainable Neural Network for Spotting Text with Arbitrary Shapes

2.3.1. Summary

Similar to (Bartz et al., 2018), (Yang et al., 2019) notes that text detection and recognition handled separately lead to sub-optimal performances because both tasks are highly correlated. Therefore they propose a text spotter which both detects and recognizes arbitrary text. (Arbitrary just being text found in real world settings). Another commonality between these paper and (Bartz et al., 2018) is that the task of recognition and detection can be done end-to-end.

Diving deeper into the methodology the name of their whole architecture is called Mask TextSpotter. What is really special about this architecture is that unlike the architecture listed above it can handle both "regular" and "irregular" text in 2-D space. To understand how this is done there are four components that are involved: A. A Feature Pyramid for the backbone, B. a RPN that will generate text proposals, C. a Fast R-CNN that is used for 1. bound box regression,and D. a mask branch for text instance, character segmentation and text sequence recognition.

A. Because text in the "wild" are in different sizes,

they have to all be formatted in such a way that they are comparable. Like (Lyu et al., 2018), (Yang et al., 2019) applies a FPN backbone along with the ResNet-50. (To see why this is a good component to implement, refer back to (Lyu et al., 2018) summary to understand). But to reiterate, it improves accuracy and has little cost.

B. RPN stands for Region Proposal Network. It is the backbone for the Fast R-CNN and on of the networks that are part of it (Karmarkar, 2018). The main purpose is to create "proposals" or a rough estimate of where certain objects may lay in the image (in our case whats the rough estimate of where the text is in the image). This is helpful as it can reduce the amount of time needed for the subsequent part of the Fast R-CNN.

C. Now that the RPN has figured out "proposals" for where the text lies and gives rough estimates through bounding boxes, the Fast R-CNN then includes the classification task and the regression task where both tasks will output much more accurate bounding boxes for detection.

D. The role of actually detecting and recognizing the text is the mask branch. There are three parts: text instance segmentation task, character segmentation task, and text sequence recognition. The character and text instance segmentation is easy enough to deal with and a feature maps are created. However, there are some limitations with these maps and therefore, the authors proposes a SAM (spatial attention module). It decodes the text sequence from the feature map. This module will produce a better representation of the various shapes.

In total, the Mask TextSpotter is a comprised of four components that is not only very easy to train, it can work on regular or irregular text.

2.3.2. Critique

This architecture does a very good job of actually spotting and recognizing the images. The data sets that they test their architecture on is the same as the data sets in the (Lyu et al., 2018) article. And I have to say that I am pretty impressed. They have accuracys as high as 99.8. When comparing (Lyu et al., 2018) and (Yang et al., 2019) results, (Yang et al., 2019) does better in every data set. However, one drawback for this architecture is that it is a little slower. However, the speed of the model is still comparable to other state-of-the-art architectures. Most architectures have an FPS between 0-6.9. However, (Yang et al., 2019) FPS is 9.0, which is definitely slower, but the results that it outputs outweighs the time it takes to detect and recognize the text in my opinion. This architecture is also much better than (Bartz et al., 2018) because it's model does not have to be trained specially. It is adaptable to any data set.

However, some questions I have is why does the SAM (spatial attentional module) do much better compared to previous methods done? The authors specify that their method does not rely on "real-world character-level annotations" which proves SAM's effectiveness. However they are not really clear how that works.

3. Implementation Goals

For the future I would like to extend the research paper SEE: Towards Semi-Supervised End-to-End Text (Bartz et al., 2018). This paper already has a github repository (located here: https://github.com/Bartzi/see) where they have a very detailed READ me. In addition, this paper implements a single deep neural network to accomplish two tasks. As an undergraduate, this might be best to work with as I do not have much experience with any of these networks, modules, or models. In their paper, (Bartz et al., 2018) claimed that their model works well in a natural setting. I was testing the text recognition model on images from the data set provide by (Bartz et al., 2018) and I saw that their recognition model worked very well. However, when I tried to run their text recognition model on the CUTE 80 data set I saw that it was not accurate at all. I realized this is because the images provided in the data set where prepossessed and rectified. However, the images from CUTE 80 where not and all the images had curved text. Therefore, as shown in Figures 1 through 4, for the word egomaniacs, the word is cropped and in a straight line, while the word crocodile is curved and in a natural setting. Figure 2 and 4 are the output given for the images egomaniacs and crocodiles respectively and one can see that the word egomaniacs is accurately interpreted, while the word crocodile is not.

Figure 1. Image with the EGOMANICS from data set provide by Bartz

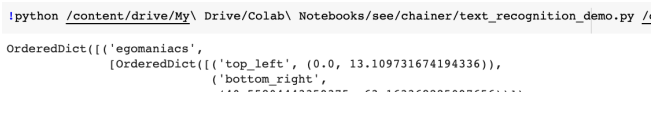

Figure 2. Output from the data set provide by Bartz correctly identifying the word

While reading (Yang et al., 2019) paper, one of their proposed future works was "to extend the proposed method to an end-to-end text recognition system which can deal with text instances of arbitrary shapes,". Bartz model is an end-to-end text recognition system and I have decided to extend their model by adding a text detection model created

Figure 3. Image from CUTE 80 with the word crocodile

```
!python /content/drive/My\ Drive/Colab\ Notebooks/see/chainer/text_recognition_demo.py

OrderedDict([('yarne',
              [OrderedDict([('top_left', (0.0, 26.544153213500977)),
                            ('bottom_right',
```

Figure 4. Output given from Bartz model not interpreting the word from CUTE80 correctly

by (Long et al., 2018) and text rectification process. (Yang et al., 2019) mentioned using TPS as a way to straighten out the image which is what I will use for this extension. This way, now that the images will be processed and can be feed into the Text Recognition model created by (Bartz et al., 2018) to achieve competitive results.

4. Methodology

The overarching plan for this extension again is to add a text detection and text rectification process before applying a text recognition model on the rectified images. Based on my research the best text detection model for curved images is (Long et al., 2018). According to (Yang et al., 2019) the best image rectification process is TPS. The text recognition model will then interpret the rectified words processed from TPS.

4.1. Text Detection Model

When looking for a text detection model I realized I need two things from the model: the TCL of a word and the contours of the words. The TCL or the text center line, is a line that is created by the center of each character of a text instance and contours of the word is a line that encapsulates the text instance. (Long et al., 2018) provided all of these inputs. In more detail the text center line created by first detecting the center of each character. The center is then surrounded by a disk. Then the center of the disks are used to create a line that string through each center and create the text center line. The contours is the line that encapsulates the entire word. Fig. 5 depicts what this looks like. The difference with (Long et al., 2018) and previous state of the art is that they do not assume that text is in a straight line, therefore their methodology works well on curved image. To detect curved text (Long et al., 2018) uses an FCN and FPN inspired model to predict the score maps of TCL and TR (text region) where it treats the detect text or word as t. S(t) represents the word as a series of disks where each disk contains a character in the word:

$$S(t) = D_0, D_1 ... D_n$$

where each D_i has 3 geometric attributes = (c, r, θ). c is the center of disk, r is the radius of the disk and θ is the orientation of the disk. The radius is calculated by half the width of t, the orientation θ is tangential direction of the center linear around center c.

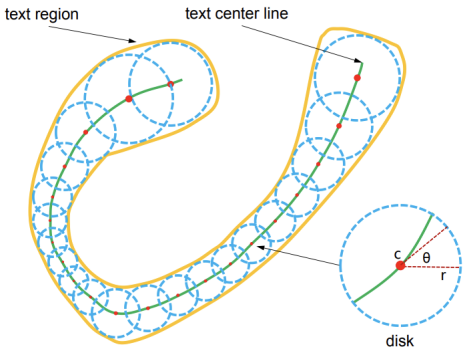

Figure 5. Example of the Text Snake or the TCL of a word given by (Long et al., 2018). The blue dotted line is the disk, the yellow solid line is the contours of the word, and the green line is the TCL

The network Architecture works consists of 5 stages of convolution where in each stage the feature map is fed to the feature merging network. Each feature map is represented by f_i where i = [1,5] and h_i represents the feature maps of merging units. After merging is complete an additional upsampling layer and 2 convolutional layers are used to receive the dense prediction. Upsampling is a process of increasing the sampling rate by adding zeros and treating them as samples received in order to maintain its length with respect to time. This way the prediction score has the same size as the input image. The prediction is given as:

$$P \in R^{hxwx7}$$

where the 4 channels are the logits (function that maps probability to R) of TR/TCL and the last 3 channels are for r, cosθ , sinθ. Because the output was the logits of TR/TCL the final predictions can be obtained by taking the softmax and regularizing cosθ and sinθ. The TCL is saved as a 2D list that consists of pairs of [x,y], where x and y are the coordinates of the each center point. This is the output I used and fed into the TPS algorithm. The TR is saved and fed in to the

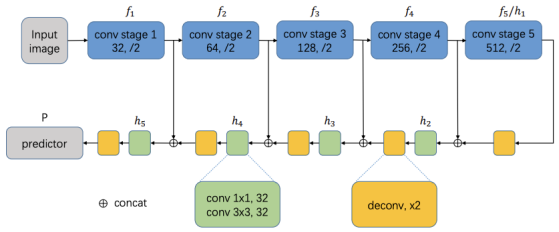

Figure 6. Network architecture created by (Long et al., 2018)

TPS algorithm as well. The github was incorporated into this project modified in order to receive the TCL. This text detection model has code written and located in this repository (https://github.com/princewang1994/TextSnake.pytorch)

4.2. TPS

TPS, also known as thin plate spline, is an interpolation method. A spline is a piece wise polynomial curve while interpolation is a process of creating new data points within the constraints of known data points. The main point of interpolation is to find the best equation for a known curve, in our case, the spline curve. The term thin plate is analogous to the bending of a thin sheet of metal which is essential what TPS does. In Fig. 7 part (a) the "thin plate" is the image and there are two important features which are the source points depicted as blue circles and the target points denoted by 5 red x's. Part (b) shows the end results of TPS where it moved the source points to their respective target points and warped the image in the process. Essentially, it is a coordinate mapping from R^2 to R^2.

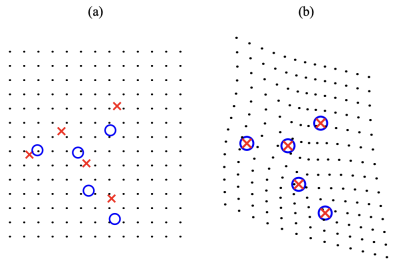

Figure 7. TPS warping process. (a represents before TPS (b represents after TPS

Looking at the algorithm in greater detail we first set the source (original) points as (x_i, y_i) and the target (end) points as v_i where *i* = [1,*p*] where *p* is the numbers of points in the data set. The first step in achieveing the shape transformation is this equation:

$$Ur_{ij} = r_{ij}^2 log(r_{ij})$$

This equation calculates the relative amount of 'energy' required to achieve a bend between (x_i, y_i) and v_i. It requires more 'energy' ie. more difficult to achieve a bend between closely spaced landmarks than between landmarks located at a certain distance from each other.

The TPS interpolant calculates the 2D representation of the 3D TPS surface which is given in the form:

$$f(x, y) = a_1 + a_x x + a_y y + \Sigma_{i=1}^{p} w_i U(\| (x_i, y_i) - (x, y) \|)$$

To ensure that $(f(x_i, y_i))$ has square integrable second derivatives two conditions must be met:

$$\Sigma_{i=1}^{p} w_i = 0$$

$$\Sigma_{i=1}^{p} w_i x_i = \Sigma_{i=1}^{p} w_i y_i = 0$$

With the two conditions, $U(r)$, and the TPS interpolant form, we can finally yield a linear system for the TPS coefficeints which is given as follows:

$$\begin{bmatrix} K & P \\ P^T & O \end{bmatrix} \begin{bmatrix} w \\ a \end{bmatrix} = \begin{bmatrix} v \\ o \end{bmatrix}$$

$K_ij = U(\| (x_i, y_i) - (x, y) \|)$. The *K* matrix represents the distances between (x_i, y_i) and v_i. The *i*th row of *P* is (1, x_i, y_i). The *P* matrix represents the distances of the coordinates (x_i, y_i). *O* is a 3 x 3 matrix of zeros, *o* is a 3x1 column vector of zeros, w and v are column vectors formed from w_i and v_i and *a* is the column vectors with elements a_1, a_x, a_y. Now the (*p*+3) x (*p*+3) matrix will be considered as L. The upper left *pxp* block of L^{-1} represented by L_p^{-1}. The TPS surface can now be calculated by inverse of matrix *L* and the target matrix denoted by X_c where the final equation is given as:

$$TPS_{uniform} = X_c L_p^{-1}$$

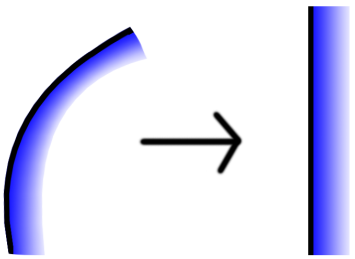

Figure 8. A Spline Curve converted into a Straight Line

$TPS_{uniform}$ represents a 'grid' that has been warped which in our case is the image itself. The (x, y) coordinates can be determined by (x_t, y_t). After understanding the algorithm, one can see that a Spline can be used to convert into a Straight line using the TPS algorithm. In order to do

this, first, I created the TPS class that followed the algorithm. (I had tried to use the openCV library for TPS transformation but it only rotated the image by a certain degree). Now I had to figure out what my source and target points were. Evidently, the source points is the TCL given from the text detection model mentioned above. However, the destination target had to be a straight line. I kept the x coordinates from the TCL line the same but set the y values of each as the half the height of the image. However, as mentioned in the algorithm one had to calculate L^{-1}. But, in this case the matrix will be singular and therefore cannot have an inverse. Therefore, in order to circumvent this issue instead of using the inverse function, I used least squares regression. This methodology does work but increases the computation time.

After applying TPS on the image, I cropped the image in a rectangular form. The height of the image is denoted as (h_t), and the contours are determined by (c_{xi}, c_{yi}) This rectangular form can be denoted by (left, top), (right, bottom). Left = min(x_t), right = max(x_t), top = max(c_{yi}) + h_t/10, bottom = max(c_{yi}) - (h_t)/10. A grey scale is then used on the images as the last step.

4.3. Text Recognition Model

The Text Recognition model used to determine the word was created by (Bartz et al., 2018). The Text Detection Model used by Bartz used a spatial transformer to detect images. The STN produces a set of *N* regions. In this methodology however, the text detection is replaced by (Long et al., 2018) and the text recognition model reads from the images in the processed CUTE80 data set. Bartz uses a CNN model that predicts the probability distribution labeled at over the label space which is computed as A$\cup\{\epsilon\}$. A is the alphabet being used for recognition (in our case the English alphabet) and ϵ represents the blank label. This network was previously trained by running a LSTM. An LSTM is a RNN that stands for long short-term memory where gates are used to retain or throw information that is deemed important or irrelevant respectively. It was trained on a fixed number of time steps which is essentially the number of characters. The maximum numebr of characters this model was trained on was 23. In the case of using CUTE80, this is perfectly acceptable as no single text or word in these images have more than 23 characters.

5. Evaluation and Results

I evaluated my method on one data set which was the CUTE 80 data set created by (Anhar Risnumawana).

CUTE80 (CUTE) is designed to evaluate curved text recognition. It has 80 images for testing.

When testing the text rectification part, I realized that the text recognition model can only handle one word at a time, so even if the text detection model could detect multiple words, I took the first word it detected and focused on rectifying that specific text instance. In order to give a quantitative analysis of how accurately the text recognition model determines the word I created a ground truth file for all the first words detected in each image. Then I compared each character in the ground truth file word with the each character in the predicted word. Each time a character is correct I added one to a sum. Then I divided that sum by the number of characters present in the ground truth word. Then I repeated this process for every word in the data set. Once all the accuracies for each word was calculated, I calculated the overall accuracy by dividing the sum of these accuracies by the total number of words. This is similar to how (Bartz et al., 2018) calculated their accuracies on the FSNS and SVHN data set.

As seen in Fig. 9, 10, 11, after the process of text detection and text rectification, the text recognition model could accurately determine the text in the image.

Now, with the data set given by (Bartz et al., 2018), we can see that their model works very well and achieved an accuracy score of .96. However, when I ran the evaluation script on the CUTE 80 data set it achieved an accuracy score of zero. After running the the model on the processed CUTE80 data set after TPS, however, we can see the accuracy has jumped to .42 seen in the Sequence Accuracy Table below. While 42% is not a competitive result, it shows how much of a difference it makes to perform TPS on an image before feeding it into a text recognition model. It shows the importance of preprocessing data sets before using them.

Sequence Accuracy	
Data set provided	96.0587%
CUTE80	0.0000%
Post TPS CUTE80	**42.0833**%

This table shows the accuracies of each data set with Bartz's text recognition model

Figure 9. Image from CUTE 80. The left picture is the image with TCL line that has been detected. The right picture shows the TPS process and straightening the image

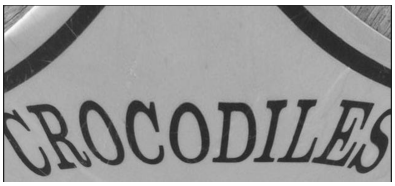

Figure 10. CUTE 80 image processed for text recognition model

```
!python /content/drive/My\ Drive/Colab\ Notebooks/see/chainer/text_recognition_demo.py
OrderedDict([('crocodiles',
              [OrderedDict([('top_left', (0.0, 27.86699104309082)),
                            ('bottom_right',
```

Figure 11. Bartz text recognition model successfully works in interpreting crocodile

6. Limitations and Challenges

6.1. Limitations

As shown in Fig. 12 and Fig. 13 , when an image is curved too much, the TPS algorithm does not work as well. One reason is that there is a congregation of points in a similar area rather than having them spread out. This biases the algorithm and warps the image towards that direction rather than evenly changing the image. Mathematically, as explained in the TPS section, the amount of difficulty in mapping points closer to each other is raised significantly. As one can see in Figure 12 there are more clusters of points on both ends of the TCL. Therefore the image is skewed towards those clusters more. In the future, in order to fix this issue, there should be an algorithm that takes points from the TCL that are more evenly distributed. This way there is less difficulty when performing the TPS method.

Another limitation is that the algorithm I created is very slow. The TPS algorithm itself has a runtime of $O(n^3)$. Therefore, when I was working with images that were more than 2000 x 2000 pixels the algorithm took a couple of minutes to warp the image. I had to separate out the processes of text detecting, warping, and recognition into three parts. Running each of these processes separately was time consuming.

Figure 12. TPS transformation on Starbucks sign

```
!python /content/drive/My\ Drive/Colab\ Notebooks/see/chainer/text_recognition_demo.py
OrderedDict([('iarbucs',
```

Figure 13. Bartz text recognition model does not correctly identify Starbucks

6.2. Challenges

There were two main challenges when creating this project. One was the allocation of time given to me for GPU in Google Collab. After running my scripts on the GPU for 15 minutes, the active session would crash and I was disconnected from using the GPU for the next 8-12 hours. This was very frustrating but opened my eyes to the fact that when creating scripts I cannot always change a little bit of my code and then run it to test my code. I had to really think about the changes I was making and then test it out once I was sure it was working. The second challenge in this project was working with data sets. When I tried working with the FSNS and SVHN data sets I under estimated their size and realized I had no space on my Google Drive to work with these data sets. Therefore, I had to find a data set that was some what smaller. This way I had a lot more control over what was happening. However, the extra step was that I had to create a ground truth file for each of the images in order to evaluate my methodology.

7. Take Home Message and Future Goals

Betwen STN and TPS, it seems that TPS has faired better and does a better job of handling both straight and curved text, while an STN can only handle relatively straight text. We can see that there was a significant increase in the accuracies when the images were processed. The obvious goal was to try to achieve an accuracy that was equivalent to or greater than the accuracy on the data set provided, but again the data set provided was from images that were original straight, and processed by the STN. However, the STN did not help with the curved text in the CUTE80 data set.

For the future, one can see that the TPS algorithm does not work well when the TCL is not evenly distributed. This is most likely because every point is changed from its original point to its destination point. If there are multiple points in a certain area, the TPS algorithm finds mapping coordinates that are closer to each other more difficulty or requires more ‘energy’. Furthermore, there are multiple ways to decrease the TPS algorithm computation, one such way is by adding a regularizer. In the future, I hope to create a much more robust TPS algorithm incorporating this method.

References

Interpolation methods in computer graphics. *Geeks for Geeks*, May 2020. URL https://www.geeksforgeeks.org/interpolation-methods-in-computer-graphics/.

Anhar Risnumawana, Palaiahankote Shivakumara, C. S. C. C. L. T. A robust arbitrary text detection system for natural scene images. URL https://doi.org/10.1016/j.eswa.2014.07.008.

Bartz, C., Yang, H., and Meinel, C. In *SEE: Towards Semi-Supervised End-to-End Scene Text Recognition*, 2017.

Bartz, C., Yang, H., and Meinel, C. See: towards semi-supervised end-to-end scene text recognition. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/viewFile/16270/16248.

Donato G., B. S. Approximate thin plate spline mappings. In *Proceedings of the European Conference on Computer Vision (ECCV)*, May 2002. URL https://doi.org/10.1007/3-540-47977-5_2.

Elonen, J. Thin plate spline editor - an example program in c++. 2005. URL https://elonen.iki.fi/code/tpsdemo/.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. Mask r-cnn, 2018. URL https://arxiv.org/pdf/1703.06870.pdf.

Karmarkar, T. Region proposal network (rpn) — backbone of faster r-cnn. *Medium*, Aug 2018. URL https://medium.com/egen/region-proposal-network-rpn-backbone-of-faster-r-cnn-4a744a38d7f9.

Keller, W., B. A. Thin plate spline interpolation, Feb 2019. URL https://link.springer.com/article/10.1007/s00190-019-01240-2.

Long, S., Ruan, J., Zhang, W., He, X., Wu, W., and Yao, C. Textsnake: A flexible representation for detecting text of arbitrary shapes. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018. URL https://openaccess.thecvf.com/content_ECCV_2018/html/Shangbang_Long_TextSnake_A_Flexible_ECCV_2018_paper.html.

Lyu, P., Liao, M., Yao, C., Wu, W., and Bai, X. Mask textspotter: An end-to-end trainable neural network for spotting text with arbitrary shapes. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018. URL https://openaccess.thecvf.com/content_ECCV_2018/papers/Pengyuan_Lyu_Mask_TextSpotter_An_ECCV_2018_paper.pdf.

MacLeod, N. Shape models ii: The thin plate spline. 2009. URL https://www.palass.org/sites/default/files/media/palaeomath_101/article_19/article_19.pdf.

Phi, M. Illustrated guide to lstm's and gru's: A step by step explanation. *towards data science*, Sep 2018. URL https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.

princewang1994. Textsnake.pytorch. https://github.com/princewang1994/TextSnake.pytorch, 2019.

Shi, B., Wang, X., Lyu, P., Yao, C., and Bai, X. Robust scene text recognition with automatic rectification, 2016.

Sønderby, S. K., Sønderby, C. K., Maaløe, L., and Winther, O. Recurrent spatial transformer networks, 2015. URL https://arxiv.org/pdf/1509.05329.pdf.

Tsang, S.-H. Review: Stn — spatial transformer network (image classification). *Medium*, Jan 2019. URL https://towardsdatascience.com/review-stn-spatial-transformer-network-image-classification-d3cbd98a70aa.

WarBean. tps stn pytorch. https://github.com/WarBean/tps_stn_pytorch, 2018.

Yang, M., Guan, Y., Liao, M., He, X., Bian, K., Bai, S., Yao, C., and Bai, X. Symmetry-constrained rectification network for scene text recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019. URL https://openaccess.thecvf.com/content_ICCV_2019/papers/Yang_Symmetry-Constrained_Rectification_Network_for_Scene_Text_Rec

Zhan, F. and Lu, S. Esir: End-to-end scene text recognition via iterative image rectification, 2019. URL https://arxiv.org/pdf/1812.05824.pdf.