Assisting Handwriting via Text Recognition and Auto-Completion on Device

Hidehisa Arai
TUM Department of Informatics, Chair for Applied Software Engineering
Technical University of Munich
Munich, Germany
koukyo1213@g.ecc.u-tokyo.ac.jp

***Abstract*—Handwriting is still in strong demand as a way of communication and saving knowledge even in the era of new technology. Although writing sentences is easier with a keyboard, handwriting is better at expressing illustrations and formulas. It is also attractive that the degree of freedom of writing is larger than keyboard input. Despite these advantages, keyboard input is widely used in computer-based processing because handwriting input is so diverse that it is not in a format suitable for computer processing. The major drawbacks of handwriting input resulting from this characteristic are its low portability and little assistance in inputting. To address these drawbacks, Handwriting Recognition, technology which converts handwritten input into a form that can be handled by a computer, has been studied for a long time. Driven by the rise of electronic pens and tablets, we have more opportunities than before to write by hand with these devices. Therefore, the need for Handwriting Recognition on electronic devices is also increasing recently. However, Handwriting Recognition applications on devices are still evolving and the true value of electronic tablets as a medium of handwriting has not yet been exploited enough. This report focuses on application of Handwriting Recognition on iOS device and proposes auto-completion using the recognition result as an assistance for handwriting input to alleviate the inconvenience of handwriting. We show that our auto-completion can reduce the number of characters that need to be actually entered to write the word the user want to write, and thus reduce the lack of input assistance. We also propose a technique to significantly reduce the amount of computation required, that is also an important aspect of processing on device. We provide two metrics to measure the performance of auto-completion, and compare the performance difference between the case using the text recognition API provided by iOS and the case using the Handwriting Recognition model designed to reduce the amount of computation.**

I. Introduction

Handwriting has played a very important role as a means of human knowledge preservation or communication from the invention of letters to the modern times. Its role has been reduced with the spread of printing and computers, but in many parts of our lives, we still have the opportunity to write and read handwriting. For example, Students in a classroom still prefer to use paper and pen or electronic tablet to store knowledge, mathematical formulas, graphs, etc., rather than taking notes using keyboard and computer. On the other hand, in computer work such as searching or document writing, input is almost always performed with keyboard, and input by handwriting is rare. This is, of course, because computer cannot handle handwritten input as text as it is, but also because keyboard is more efficient for inputting characters. With a keyboard, we can enter characters with less movement compared to handwriting. Besides, there are also assisting functions for keyboard input such as copy and paste, searching, or formatting. Since keyboard input has these overwhelming advantages, people usually try to use keyboard even for tasks which are actually easier to be done by hand, or to use keyboard for writing and use tablet device for those tasks that are not achievable with keyboard. However, with the spread of tablet devices in recent years, this situation is showing signs of change. On these devices, input work that keyboards are not good at, such as drawing illustrations and writing mathematical formulas, can be easier. In addition, applications [1][2] have appeared that can handle handwriting as text format by applying Handwriting Recognition technology that has been studied for many years. Thanks to these technological improvements, handwriting is becoming an available option for inputting text, but it is not yet as efficient as a keyboard is. One of the reason for this is the lack of input assistance. Therefore, in this work, we propose an application that combines Handwriting Recognition and auto-completion to assist handwriting and to enable it be treated as text format on devices.

Our application consists of two elements: Handwriting Recognition and auto-completion. Handwriting Recognition (HWR) is a technology that interprets handwritten inputs from sources such as papers, photographs, electronic tablets, and other devices into a format that can be easily handled by computers. In most of the modern work of HWR, the process pipeline is divided into Handwritten Text Detection and Handwritten Text Recognition [1]. The processing of text detection tends to be complicated [1], and it takes a considerable amount of time to perform processing on the entire input image. It is known [3] that longer processing time in applications that require a quick response, such as auto-completion, can seriously degrade the user experience. Therefore we propose a method to avoid performing Text Detection using a simple heuristic: Region of Interest (ROI) detection.

[1]https://www.nebo.app/
[2]http://mazec.jp/
[3]http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html

Auto-completion is a program that predicts the rest of the word a user is typing based on the letters entered or the recent word pairs [2]. It is widely used in web browsers, in email programs, in source code editors or in word processors. If the correct word is included in the predicted words list, the user can save the time to enter the rest of the word. In this work, to make the process simple, we used forward matching algorithm for this purpose.

In the following section, typical approaches of HWR and related works are introduced. Section III provides technical details of the approach to the problem addressed in this report. Section IV describes the result of the approach and discuss on that. Section V concludes this report with future prospects.

II. Related Work

A. Handwriting Recognition Taxonomy

HWR can be roughly devided into two appproaches: online approach and offline approach [3]. While online approach uses information on the trajectory of the pen tip obtained from a special pen for classification, offline method uses optically scanned images as input and performs recognition using computer vision techniques. In this work, for simplisity of the pipeline, we focuses only on offline approach.

Offline approach of HWR can be positioned as one variant of Scene Text Detection / Recognition, which is a technology to extract and recognize text information written in natural images. Due to the recent development of Neural Networks technology, much research has been done in this field to this day. Except for few methods [4] [5], most approaches of Scene Text Detection / Recognition separate text detection and text recognition and perform stepwise inference.

B. Detection

Scene Text Detection can be subsumed under general object detection, therefore those methods usually follow the same procedure of object detection, which is dichotomized as one-stage methods and two-stage ones [6]. After the emergence of FasterR-CNN [7], most of the modern text detection algorithms are based on FasterR-CNN, YOLO [8], SSD [9].

In addition to the general object detection model, text detection models are devised to detect tilted bounding boxes [10] [11] and character regions of arbitrary shapes [12], or to simplify the pipeline [13], since pipeline of text detection tends to be complicated [6].

C. Recognition

Some text recognition algorithms devide the task into character segmentation and character recognition [14] [15]. Character segmentation is considered as the most challenging part of scene text recognition, and may affect overall accuracy. It is especially difficult to segment connected characters such as cursive. Therefore some techniques which do not rely on character segmentation have been developped so far. This report introduces a method called Connectionist Temporal Classification (CTC) [16].

CTC was first proposed to handle sequence labeling of arbitrary length, requiring no pre-segmented training data. A CTC network outputs probabilities for each label at each time step. Time step length can be any length longer than label length. The output at each time step is the probability of the classess to be recognized plus the extra class representing "blank". Let this output probabilities be $\mathbf{y} = (y_1, y_2, \cdots, y_w)$ and denote by y_k^t the activation of label k at time step t. Given this probability distribution, the conditional probability of the sequence is calculated as follows.

$$p(\pi|\mathbf{y}) = \prod_{t=1}^{w} y_{\pi_t}^t \tag{1}$$

Then a many-to-one mapping $\mathcal{B}$ is defined to transform the sequence π to a shorter sequence. The final predicted label is obtained by this mapping. This mapping removes all blanks and repeated continuous labels from the sequence. For example, $\mathcal{B}$ maps the predicted sequence "aa-p-pl—-ee" to "apple", where "-" represents the "blank". Since this mapping is many-to-one mapping, different sequences may be mapped to the same sequence. Therefore the probability of the final output sequence is the sum of all possible conditional probabilities of all π corresponding to that final sequence.

$$p(l|\mathbf{y}) = \sum_{\pi} p(\pi|\mathbf{y}) \tag{2}$$

where π represents all π which produces $l = \mathcal{B}(\pi)$.

The output of the classifier should be the most probable labeling for the input sequence.

$$h(\mathbf{y}) = \arg\max p(l|\mathbf{y}) \tag{3}$$

In general, there are a large number of mapping paths for a give sequence, thus calculation of $\arg\max$ requires heavy computation. In practice, following two approximate methods are known to give us a good result.

The first method is based on the assumption that the most probable path can be approximated by the sequence of most probable labeling

$$h(\mathbf{y}) \approx \mathcal{B}(\pi^*) \tag{4}$$

where π^* is a set of labels which get the highest probabilities at each time step. Although it works well, it is not guaranteed to get the most probable labeling.

The second method is to use forward-backward algorithm to efficiently search for the most probable sequence. With enough time, this approach can always find the most probable labeling from the input sequence, but the amount of computation increases exponentially with respect to the sequence length, it is not practical to find the exact solution.

To train the network with the dataset $\mathcal{D} = \{I_i, l_i\}$, where I_i represents the input image and l_i represents the corresponding label, maximum likelihood approach it utilized. The objective function of this can be negative log-likelihood

$$\mathcal{O} = -\sum_{(I_i, l_i) \in \mathcal{D}} \log p(l_i | \mathbf{y}_i) \tag{5}$$

where $\mathbf{y}_i = f(I_i)$ and $f(\cdot)$ represents the classifier. To minimize negative log-likelihood, Stochastic Gradient Descent (SGD) can be used. To summarize, a model that performs text recognition with CTC can be obtained by defining a network that outputs a sequence longer than the required label length and training the network so as to minimize the loss function (5). After the fitting, model outputs a sequence of probabilities of labels for a given input. Each time step of the sequence corresponds to, if the input is an image, image patches arranged in the direction of writing. We then get final prediction by putting the output sequence to a many-to-one mapping. There are several options for this many-to-one mapping, but the one to get highest probabilities at each time step and the one which use forward-backward algorithm is considered to achieve good performance in practice.

III. Methods

As in the case of Scene Text Detection / Recognition, it is effective to separate Text Detection and Text Recognition and treat them as different problems. Furthermore, it is possible to record the written area due to the characteristics of the electronic tablet, so using a simple heuristic on the trajectory data eliminates the need to actually perform text detection. The role of this step, namely "Region of Interest Detection" step is to reduce the size of data passed to the subsequent processing and suppress the increase in the amount of calculation. Detected regions are then preprocessed and passed to the text recognition module. In the text recognition module, two patterns of recognition using CTC and recognition using the API provided by Apple were verified.

A. Region of Interest Detection

The purpose of Region of Interest (ROI) detection is to cut out sub-sequence representing words and illustrations from the dot sequence of the pen tip trajectory and cut back the data to be passed to the subsequent processing to reduce the amount of calculation and improve the recognition accuracy. Region of interest is a region including a dot sequence of the word or the illustration. To specify the region, two assumptions are made on region of interest.

1) Objects drawn in the region of interest are close in time when drawn
2) Objects drawn in the region of interest are spatially close

Based on these assumptions, ROI is detected by the following algorithm 1. Then the smallest rectangle containing the obtained point sequence is the ROI. It should be noted that we prioritize the reduction of the amount of computation and terminate the search for subsequences on the way.As a result, points that are not continuous in time but close in space may not be included in the ROI depending on the threshold setting. However, when priority is given to spatial proximity, it is necessary to parse the entire trajectory every time we perform ROI detection, and if the size of the point sequence becomes large, it may be a bottleneck in calculation. Therefore we decided to use this method, but better approaches will be discussed in the section V.

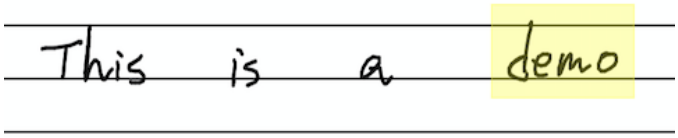

Fig. 1. The region with yellow overlay is detected region of interest

Algorithm 1: ROI detection

Input: $S = \{s_1, s_2, \cdots, s_n\}$ *where* $s_i = (c_i, t_i)$
Output: $\sigma = \{s_k, s_k + 1, \cdots, s_n\}$ $1 \leq k \leq n$

subset := $\{\}$
$c_{prev} := c_n$
$t_{prev} := t_n$
for s_i *in* $reversed(S) \setminus s_n$ **do**
  $c_i, t_i \leftarrow s_i$
  if $t_{prev} - t_i < t_{threshold}$ **then**
    $subset \leftarrow subset \cup \{s_i\}$
  else
    if $distance(c_{prev}, c_i) < c_{threshold}$ **then**
      $subset \leftarrow subset \cup \{s_i\}$
    else
      break
  $c_{prev} \leftarrow c_i$
  $t_{prev} \leftarrow t_i$

Figure 1 shows an example of the region detected with this heuristic.

There are some cases where such heuristics do not work. For example, when the scale of the depicted object is large, the gap between the sequence of points constituting the object may be too large and fail.It is also a major problem that temporal and spatial proximity depends on parameters and sometimes does not match intuition. However, it is certain that this method can narrow down the target area for text recognition with a very small amount of computation, so this report adopted this method.

B. Recognition

In text recognition, two models were tried: a model that directly reads the content from the result of region of interest detection using CTC, and the model which is provided by Apple[4]. Note that Apple's VNRecognizeTextRequest API does not disclose its implementation or the model used inside, it is unknown exactly what processing is being done.

[4]https://developer.apple.com/documentation/vision/vnrecognizetextrequest

1) CTC: CTC models are usually constructed using CNN model with Recurrent Neural Network (RNN) as the output layer. However, RNN performs computations slowly compared to CNN, and in recent years its effectiveness in CTC is sometimes questioned [17]. Given that all calculations are done on iOS, and based on research that the output layer of the CTC model can also be configured using CNN [18], in this report a model which is fully constructed only by CNN was used for CTC model. The detail of the model configuration can be found in the repository[5]. To get an approximation for most probable labeling, greedy policy, the policy to take the most probable label at each time step, was used.

2) VNRecognizeText API: VNRecognizeText API is an API provided by Apple officially, and its function is to detect text from the whole area where the drawing is and read it. Therefore there's no need to detect the region of interest to read the content inside. However, in order to make the inference fast, we used ROI detection as a preprocessing before putting an image to the API. This API is completely a black box, accepts an image of any shape as input, reads the text in it, and outputs it with position information.

VNRecognizeText API has two modes for recognition, namely '.fast' and '.accurate'[6]. According to the official introduction video[7] if VNRecognizeText API, '.fast' uses traditional feature based method inside while '.accurate' uses more sophisticated Computer Vision model inside. As the names of those imply '.fast' is faster but not so accurate while '.accurate' is slower but more accurate. Both '.fast' and '.accurate' were tested. However, since '.fast' failed to recognize any handwritten letters, measurement on '.fast' was not conducted.

C. Auto-Complete

In this report, we deal with auto-completion as an application of the inference result. This auto-completion emphasizes simplicity and presents words that start with infered result in order of frequency. To get the word candidates which start with infered result, we created an inverse index that maps the first few letters of words to a group of words with that prefix. For example, index "elbo" corresponds to word group {"elbow", "elbow's", "elbowed", "elbowing", "elbowroom", "elbowroom's", "elbows"}. Word suggestions for auto-completion are from 'wamerican' package[8] of Debian GNU/Linux. Word frequencies were counted using wikipedia-word-frequency[9]. We implemented auto-completion to be performed each time the pen is released from the tablet surface.

D. Dataset

Handwritten character recognition and handwritten sentence recognition are fields that have been studied for a long time, so there are many data sets, but these are often provided in different formats, and there is some difficulty in eliminating differences between formats and using them for training dataset. We therefore took an approach to create a composite dataset by embedding a combination of existing handwritten-like fonts with randomly picked fontsize and randomly selected English words in the image. The words are selected from 'wamerican' package and 10,000 images were generated for training. We split the dataset to 80% / 20% each for training / validation. The fonts used are listed up in the repository[10]. Figure 2 shows an example of training data generated with this method.

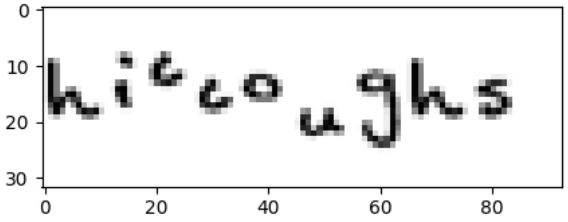

Fig. 2. Generated image using handwritten-like fonts

For image pre-processing, we only used vertical and horizontal random offset of characters in a word. Since the image taken from the prototype application has white background with no blurriness, we didn't apply further image augmentation techniques.

E. Implementation

We used Python3.6[11] and TensorFlow2.1[12] for building machine learning models and training, and used coremltools3.2[13] to convert the models into the format that can work on iOS.For training, Adam optimizer with learning rate 5e-5. 'beta_1' and 'beta_2' parameters are set to 0.9 and 0.999 each. The trainig was performed 50 epochs with batch size 32.

To get handwritten document image from the iOS electronic tablet, a function to print the input from the Apple Pencil[14] on the white canvas according to the path of the pen has been implemented.

For training the model, Google Colaboratory environment [15] has been used. All the implementation is published at the GitHub repository[16].

IV. Results & Discussion

A. Region of Interest Detection

It is desirable for ROI detection to cut out the region which is focused by the user. For example, when a user is writing a sentence,ROI should be a word written, and when a user is

[5]https://github.com/koukyo1994/iOS-note-v2/blob/master/prototype/py/model.py
[6]https://developer.apple.com/documentation/vision/vnrequesttextrecognitionlevel
[7]https://developer.apple.com/videos/play/wwdc2019/234/
[8]https://packages.debian.org/search?keywords=wamerican
[9]https://github.com/IlyaSemenov/wikipedia-word-frequency
[10]https://github.com/koukyo1994/iOS-note-v2/blob/master/prototype/py/fonts/list.txt
[11]https://www.python.org/downloads/release/python-360/
[12]https://www.tensorflow.org/
[13]https://apple.github.io/coremltools/
[14]https://www.apple.com/apple-pencil/
[15]https://colab.research.google.com/notebooks/welcome.ipynb
[16]https://github.com/koukyo1994/iOS-note-v2

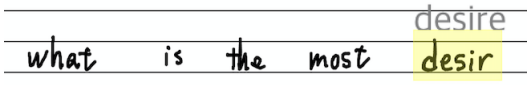

Fig. 3. ROI detection works well when cutting out a word which is being written

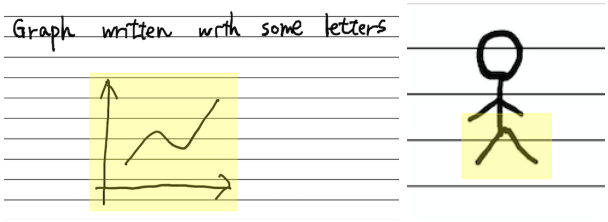

Fig. 4. Left: example of the case ROI detection worked well on illustration. Right: example of failure case of ROI detection

drawing an illustration, ROI should be the whole illustration drawn.

Since it is difficult to quantitatively measure the goodness of the ROI detection, only qualitative evaluation was performed this time. When cutting out only the words that the user is writing in the text, ROI detection worked almost without failure (Fig 3).

On the other hand, when cutting out a handwritten illustration, only a part of the illustration may be cut out if the lines constituting the illustration are not spatially close to each other (Fig 4). Note that the failure example may seem spatially close inside, but it is not the case when we split it up into a set of strokes and compare the distance between starting point and ending point of strokes.

B. Evaluation of Text recognition

In the previous section, two patterns of text recognition were tried: recognition using CTC and recognition using VNRecognizeText API. Since this report worked on recognition of handwritten text and made use of the result for auto-complete, following metrics were designed to measure the goodness of auto-completion.

- Omitted Characters Count (OCC) - The gap between how many characters did the writer actually write before he found the word he wanted to write within top 10 of the auto-completion candidates and the number of characters in the word. If it is not found after writing to the end, the score is 0. Higher value of this metric indicates better result.
- Cumulative Time for Inference (CTI) - Cumulative time spent on auto-completion until the word that the writer actually wants to write is included in the top 10 auto-completion candidates. If it is not found after writing to the end, the score cumulative time spent on the recognition at each step. Note that each time writer releases the pen tip from the tablet, recognition process runs. Lower value of this metric indicates better result.

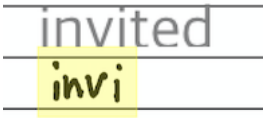

Fig. 5. Example of auto-completion showing the top 1 candidate

Fig. 6. Example of unstable inference

100 words were randomly selected from the word list of ‘wamerican‘ package of Debian GNU/Linux for evaluation. The selected word list is in the repository. Table I shows the performance of both methods on auto-completion. While OCC measure of VNRecognizeText API show better result than CTC, cumulative time elapsed for inference is almost doubled compared to that of CTC’s.

method	OCC (mean)	CTI (mean)
CTC	1.0102	1.1060
VNRecognizeText (.accurate)	3.3405	1.9751

TABLE I
PERFROMANCE OF CTC AND VNRECOGNIZETEXT API ON AUTO-COMPLETION.

Figure 5 show an example of how auto-completion works.

C. Discussion

The VNRecognizeText API is superior to CTC in character recognition accuracy, but in terms of speed it takes about three times longer to infer. Since auto-completion is an application that requires real-time performance, a small lag does not provide a good user experience. Therefore, poor performance in speed can be a problem.

On the other hand, the CTC model shows inference speed that does not the user feel uncomfortable in actual use, but the inference result is unstable, and a slight difference in notation greatly affects the inference result. Figure 6 shows an example of unstable inference.

This is probably because training of the CTC model seems to be over-fitty and strongly depends on the dataset used for training. Therefore, future tasks include reducing the reliance on datasets and using larger datasets or using data augmentation techniques to improve generalization performance.

V. FUTURE WORK & CONCLUSION

The goal of this report is to create an application to recognize sentences on iOS for documents where handwritten illustrations and characters were mixed. Although it is imperfect, the heuristic used for region of interest detection successfully detect a word out of a sentence, and can cut

handwritten illustration out from the other part of document without requreing large amount of computation. On the other hand, text recognition had a trade-off between speed and accuracy.

Future tasks include improving the accuracy of text recognition without slowing it down. Another example of future work is to perform more accurate text recognition and ROI detection using an online method. In auto-completion, prediction using information on surrounding words, and refinement of the method of presenting complement candidates can also be one of future works.

REFERENCES

[1] S. Long, X. He, and C. Yao, "Scene text detection and recognition: The deep learning era," 2018.

[2] J. J. Darragh, I. H. Witten, and M. L. James, "The reactive keyboard: A predictive typing aid," *Computer*, vol. 23, no. 11, pp. 41–49, 1990.

[3] R. Plamondon and S. N. Srihari, "Online and off-line handwriting recognition: a comprehensive survey," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 1, pp. 63–84, 2000.

[4] X. Liu, D. Liang, S. Yan, D. Chen, Y. Qiao, and J. Yan, "Fots: Fast oriented text spotting with a unified network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 5676–5685.

[5] P. Lyu, M. Liao, C. Yao, W. Wu, and X. Bai, "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)*, 2018, pp. 67–83.

[6] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, "Deep learning for generic object detection: A survey," *arXiv preprint arXiv:1809.02165*, 2018.

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

[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[9] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[10] X. Zhou, C. Yao, H. Wen, Y. Wang, S. Zhou, W. He, and J. Liang, "East: an efficient and accurate scene text detector," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 5551–5560.

[11] 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," *arXiv preprint arXiv:1706.09579*, 2017.

[12] C. Zhang, B. Liang, Z. Huang, M. En, J. Han, E. Ding, and X. Ding, "Look more than once: An accurate detector for text of arbitrary shapes," 2019.

[13] P. He, W. Huang, T. He, Q. Zhu, Y. Qiao, and X. Li, "Single shot text detector with regional attention," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 3047–3055.

[14] A. Bissacco, M. Cummins, Y. Netzer, and H. Neven, "Photoocr: Reading text in uncontrolled conditions," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 785–792.

[15] T. Q. Phan, P. Shivakumara, B. Su, and C. L. Tan, "A gradient vector flow-based method for video character segmentation," in *2011 International Conference on Document Analysis and Recognition*. IEEE, 2011, pp. 1024–1028.

[16] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 369–376.

[17] J. Puigcerver, "Are multidimensional recurrent layers really necessary for handwritten text recognition?" in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, vol. 1. IEEE, 2017, pp. 67–72.

[18] Y. Gao, Y. Chen, J. Wang, and H. Lu, "Reading scene text with attention convolutional sequence modeling," *arXiv preprint arXiv:1709.04303*, 2017.