Intelligent Time-Stamp Detection and Recognition Using an Adaptive Sliding Window Approach

Siddarth Narasimhan, Wei Huang, Niu Zheng
Ministry of Transportation (MTO)
Systems Analysis and Forecasting Office
Integrated Policy and Planning
Toronto, Canada

***Abstract*—Text is often overlaid across images and video by highway cameras, and can easily become a useful source of information. This paper introduces a novel timestamp detection algorithm, which achieves localization using a custom trained YOLOv3 model and detection using an adaptive sliding window approach and autoencoding. The algorithm has been shown to work consistently and with high accuracy across various fonts and text sizes with an accuracy of more than 90%, making it a suitable framework to be applied in numerous optical character recognition applications.**

***Index Terms*—timestamp recognition/detection, YOLOv3, autoencoders, sliding window**

I. Introduction

Cameras place text, typically in the form of timestamps, over images or video as a means of keeping track of the date and time the image was captured. While camera EXIF (Exchangeable Image File Format) data is an alternate source this image metadata, older cameras often do not make this available to the user. Given a large number of images to extract timestamps from, the manual conversion of text in images to machine-encoded text becomes a very daunting task.

Optical character recognition (OCR) is a widely explored topic in academia that can help achieve timestamp detection and recognition, however, the search for an algorithm that can generalize to every situation is still an active area of research. Timestamps are often placed in a consistent area on the image (very top or bottom) with a distinct colour, however, detection and recognition still remains an unobvious task. Some of the challenges include being able to identify text in a variety of font types and sizes and the issue that variable background noise in the image can blend with the text colour making them difficult to separate.

There are several methods that have been proposed to achieve this, but come with a few limitations. We can divide these methods into open-source OCRs that are available online, and ones that are proposed in literature.

PyTesseract and the EAST Text Detector are standard open source OCRs available that can read image text after applying filtering techniques such as binarization and Gaussian blurs [1] [2]. However, these methods do not perform well with large background noise and only work for a select number of fonts. Google's Vision API and Microsoft Cognitive Services are very accurate and work with large detection and recognition accuracy in a number of cases, but can be quite expensive to use long-term [3] [4].

In literature, research has been done towards developing solutions based on feature detection. Some of these features include edge detection using methods such as the stroke width detection algorithm [5], or through texture descriptors [6]. Some probabilistic approach coupled with image binariziation techniques have also been proposed which have had decent success in their applications [7] [8]. While these techniques have been shown to be effective, the major limitation in several of these methods is the fact that they are difficult to implement in practice due to their complex mathematical approaches. This can be particularly an issue for those who don't have the necessary background mathematical knowledge to follow their approach.

Our goal is to propose an algorithm that is not only accurate, but also inexpensive and relatively easy to build. The algorithm uses standard concepts that are supported, researched and document well in the machine learning community, so there is little resistance in being able to understand the approach to the algorithm. The algorithm uses a custom-trained YOLOv3 model to perform detection, and then uses a sliding window approach alongside an autoencoder and a simple multi-layer perceptron to perform digit recognition.

This paper is organized by first introducing the high-level structure of the algorithm in Section 2, then a detailed version of the algorithm is explained in Section 3. Finally, we present the results in Section 4 and conclude in Section 5.

II. The High-Level Proposed Method

See Fig. 1 for reference.

1) A YOLOv3 custom trained model will perform timestamp localization by proposing bounding boxes where a timestamp is found.
2) Crop image according to the bounding box(es) detected in the previous step. Resize cropped timestamp to a height of 32 px, maintaining aspect ratio, and convert to grayscale.
3) Apply bilateral filtering and canny edge detection to the image. Determine appropriate pixel size to use as a window for the font.

Primary Author: Siddarth Narasimhan (email: sidarceus@gmail.com) Corresponding author: Wei Huang (email: huangweibuct@gmail.com).

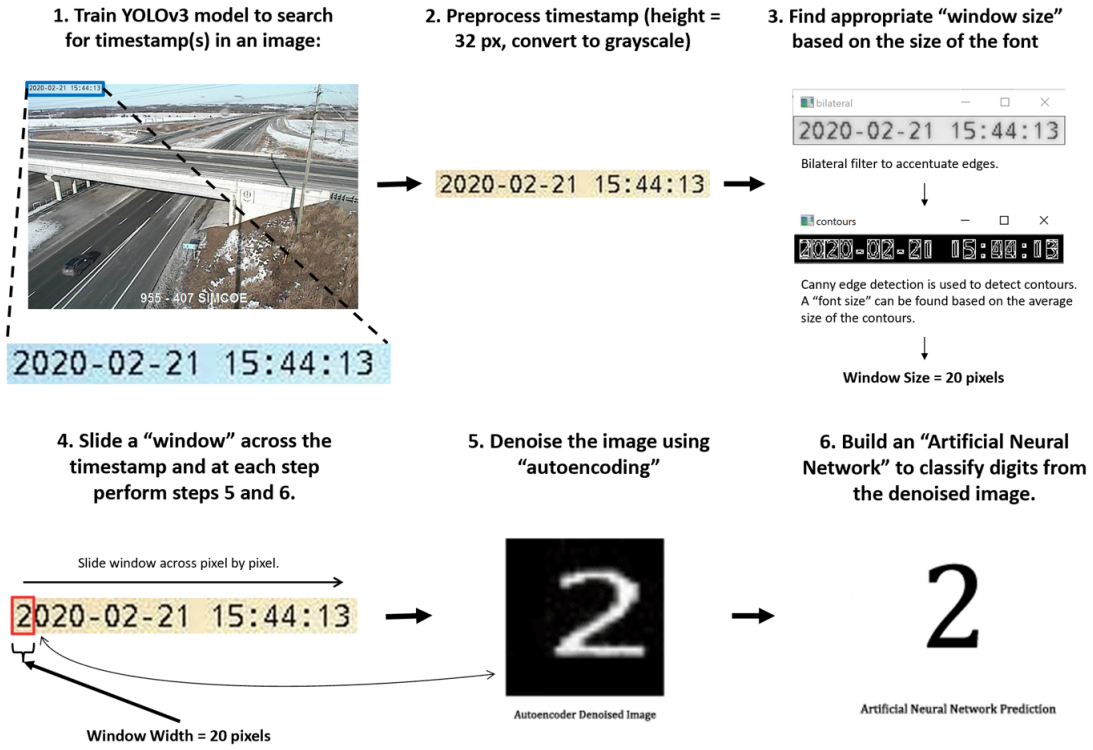

Fig. 1. High-level process depicting how a timestamp is detected by YOLOv3 and individual characters are recognized.

4) Apply the following steps until termination:
   a) Begin at the very left of the timestamp and crop window with a height of 32 px and the width found in the previous step.
   b) Resize image to 32 px $\times$ 32 px.
   c) De-noise resized image using a convolutional autoencoder.
   d) Map de-noised image to a digit using a multi-layer perceptron.
   e) Slide window incrementally by one pixel to the right. Repeat part (a).

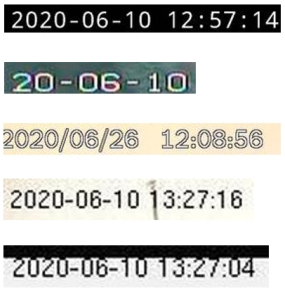

Fig. 2. Different styles of timestamps that YOLOv3 was trained on. Notice how each text style varies in font and size.

III. The Detailed Approach

A. Training YOLOv3

To train YOLOv3, a training and validation set was built consisting of 650 images and 50 images respectively, archived from 511 Ontario. Every image was manually labeled with bounding boxes using labelImg. The model was trained for 100 epochs with a initial learning size of 1e-4 an and ending learning rate of 1e-6, with a batch size of 8.

The goal of YOLOv3 is to propose bounding boxes that are centered and cropped perfectly around the time-stamp in the images. The different styles of timestamps in the dataset that the custom trained model is able to recognize is seen in Fig. 2.

B. Edge Detection

The timestamps that are cropped by YOLOv3 are resized the cropped image to a height of 32 px, maintaining aspect ratio, and then converting the image to grayscale. This is to ensure consistency across all time-stamps. This is also since the input to the convolution autoencoder in a later step accepts input images with dimensions 32 px by 32 px.

Applying bilateral filtering and canny edge detection, we can binarize the image and detect rectangular regions where characters are detection. These two steps can be achieved in OpenCV in Python. Applying a Gaussian blur accentuates edges in the image, while further removing background noise.

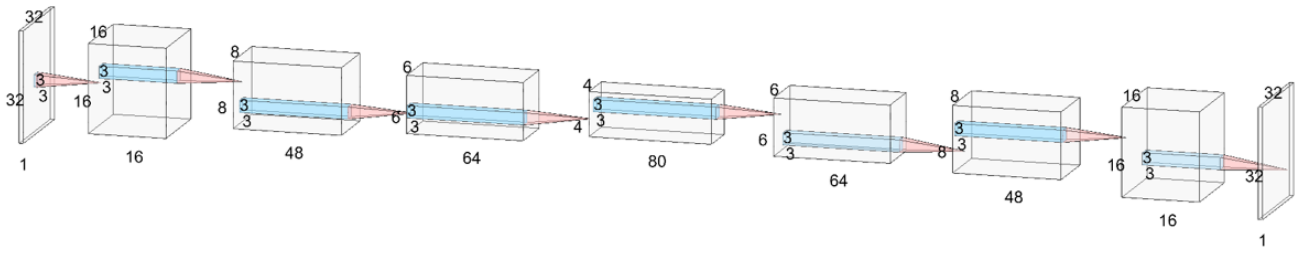

Fig. 3. Architecture for the convolutional autoencoder used to denoise images.

Canny edge detection will binarize the image after the Gaussian blur is applied and will propose detection boxes around the characters (see Step 3 in Fig. 1).

Fig. 4. Labelled Image using bounding boxes for the timestamp in the top left corner.

While text segmentation can be applied by simply cropping the image according to the detection boxes, it was noted that all the characters in the time-stamp were not always detected. Furthermore, false detections would often occur in noisy images, which can give rise to more error when performing character recognition. For this reason, we will only use the detection boxes as a means of determining the average font width of the text in pixels.

Predicting a font width size in pixels for the entire image can be done by averaging the widths of all the detection boxes subsequent to canny edge detection. However, in order to eliminate outliers in the detection, we only average the detection boxes that fall within the range of the first to the third quartile of the widths. Doing so will output an average font width size for the time-stamp, which will be useful for a later step.

C. Denoising the Image

A convolutional autoencoder can be used to denoise the cropped image from the sliding window. The autoencoder takes in a 32 px $\times$ 32 px image as input, and reconstructs the image to one of 12 images (Fig. 5). The architecture used to achieve this is shown in Fig. 3.

While autoencoders are typically trained unsupervised, the novelty of this method is to train the autoencoder using supervised learning. We create a dataset consisting of 12 different classes; the 10 digits, one to capture special characters, and another to capture spaces in between characters. This dataset consists of 55 distinct images per class that were cropped manually. The dataset was augmented four times to produce a larger dataset, which consisted of vertical and horizontal shifts to the images. During training, we feed the images into the autoencoder and we map each image to its respective denoised image. The denoised images are images that have a black background, with text in white. Sample images contained in the dataset and their corresponding mapped denoised image are shown in Fig. 5.

D. Classification with ANN

Once the image is denoised, classification becomes really simple. We can train a multi-layer perceptron (Artificial Neural Network) to do this. Any architecture to do so would suffice. Note that training a convolutional neural network to do so could also work, however, it is overkill for this application since the neural network only needs to learn the representations of 12 distinct images.

Fig. 5. The autoencoder will reconstruct the input image to one of these 12 images.

E. Interpreting the Output to an Actual Timestamp

Since a ”window” is sliding across the timestamp incrementally by pixels, there is a chance for a character to get recorded twice in the output. Since we are training the autoencoder to

not only recognize the digits but also the spaces in between the characters and the special characters (":" and "-"), we can keep track of these intermediate locations. This gives us a way to separate digits and identify duplicates in the output.

F. Additional Considerations

While sliding a window pixel by pixel is relatively computationally expensive than standard techniques, there were efforts made to apply digit segmentation directly in the image, using the stroke-width transform algorithm and OpenCV techniques. However, there was a lot of difficulty in trying to find settings for these techniques that would standardize to all different fonts, background noise and sizes. Incorrect segmentation would add additional error to the output. Also, the time to compute one timestamp ranged between 1 and 2 seconds, and this was sufficient for our application.

IV. Performance

The evaluation of timestamp localization using YOLOv3 is shown in Table I. The evaluation of timestamp digit detection using the sliding window approach is shown in Table II.

TABLE I
YOLOv3 Localization Results

Detection	Number of Images
Correct Detection	754
False Detection	5
No Detection	2
Partial Detection	17
Overall Accuracy:	**96.9%**

The testing set consists of a total of 778 images.
Correct Detection - complete detection of timestamp.
False Detection - incorrect detection of timestamp.
No Detection - detected no timestamps in the image.
Partial Detection - detection is cut off by at least 2 characters.

TABLE II
Timestamp Recognition Results

Detection	Number of Timestamps
Correct Detection	774
Acceptable Detection	177
Incorrect Detection	58
Overall Accuracy:	**94.1%**

The testing set consists of a total of 979 timestamps.
Correct Detection - complete digit recognition.
Acceptable Detection - off by one digit.
Incorrect Detection - off by more than one digit.

V. Conclusion

In this paper, we propose a framework to perform automatic timestamp text detection in images using YOLOv3, autoencoding and a sliding window approach. The algorithm has shown to be very effective and very applicable in locating and recognizing the characters in the timestamp with a wide-variety of fonts and sizes. More investigation will be required to make simplifications to the algorithm to reduce the computation time.

VI. Acknowledgements

This project was undertook by the Systems Analysis and Forecasting Office in Toronto towards with the purpose of detecting timestamps in highway camera images. A special thanks to the team and Wei and Niu for the support in developing the algorithm.

References

[1] https://www.pyimagesearch.com/2018/09/17/opencv-ocr-and-text-recognition-with-tesseract/
[2] https://www.pyimagesearch.com/2018/08/20/opencv-text-detection-east-text-detector/
[3] https://cloud.google.com/vision
[4] https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/
[5] https://www.cs.bgu.ac.il/ ben-shahar/Teaching/Computational-Vision/StudentProjects/ICBV131/ICBV-2013-1-GiliWerner/ICBV-2012-1-GiliWerner-report.pdf
[6] T. E. de Campos, B. R. Babu, and M. Varma, "Character recognition in natural images," in Proceedings of the International Conference on Computer Vision Theory and Applications, Lisbon, Portugal, February 2009.
[7] J. Weinman, E. Learned-Miller, and A. R. Hanson, "Scene text recognition using similarity and a lexicon with sparse belief propagation," in Transactions on Pattern Analysis and Machine Intelligence, vol. 31, no. 10, 2009.
[8] Bayesian Approach to Photo Time-Stamp Recognition