2023 Sixth International Symposium on Computer, Consumer and Control (IS3C) | 979-8-3503-0195-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/IS3C57901.2023.00014

A Two-Stage Pipelined Algorithm for Recognition Tasks: Using License Plate Recognition as an Example

1st Jia-Ming Yeh
Department of Computer Science and Information Engineering
National Chung Cheng University
Chiayi, Taiwan, R.O.C
6534qgas1793@gmail.com

2nd Garnett Chang
GT SPACE TECHNOLOGY CORPORATION
Hsinchu, Taiwan, R.O.C
garnett@gt-tek.com.tw

3rd Jason Lee
GT SPACE TECHNOLOGY CORPORATION
Hsinchu, Taiwan, R.O.C
jason@gt-tek.com.tw

4th Wei-Yang Lin
Department of Computer Science and Information Engineering
National Chung Cheng University
Chiayi, Taiwan, R.O.C
wylin@cs.ccu.edu.tw

***Abstract*—Although there has been a lot of research on deep learning, most of them use GPU platform to run deep network models. However, it is less desirable to utilize GPU in real-world scenarios due its relatively high cost and high power consumption. In this paper, we propose a two-stage pipelined algorithm (TSPA) suitable for the FPGA platform to avoid the above-mentioned issues. We also combine OpenCV and GStreamer so that the FPGA platform can achieve real-time performance while maintaining satisfactory accuracy. We choose license plate recognition as an example to demonstrate the feasibility of our proposed approach. We have conducted experiments using the AOLP dataset and the self-collected videos. Our proposed method achieves promising results on these videos.**

I. Introduction

Due to the rapid development of deep learning in the recent years, many image recognition technologies have been successfully deployed in a wide range of application domains. However, except for the recognition accuracy, the cost, size, and power consumption of the hardware device are also important considerations in some application scenarios. These are the challenges to be overcome to facilitate more widespread applications of deep learning technologies in our daily life. The edge computing devices provide a promising solution to the above-mentioned issues because they have lower cost, smaller size, and lower power consumption as compared to GPU.

Many research groups have explored using edge computing devices to develop deep learning applications. In [1], the authors summarize the deep-learning-enabled edge computing applications into four categories, i.e., smart multimedia, smart transportation, smart city and smart industry. Here, we place our focus of attention on license plate recognition because it is of fundamental importance to the development of smart city and intelligent transportation system. One can apply this technology to many tasks, such as street parking and highway toll collection, parking lot entry and exit control, traffic monitoring and violation detection, suspicious vehicle investigation, lost vehicle tracking, etc. Several existing works have investigated the use of edge computing devices for license plate recognition [2]–[7]. However, most of them are limited to single purpose and thus generalization to other tasks are not trivial.

In this paper, we propose a two-stage pipelined algorithm (TSPA) suitable for implementation on edge computing devices. The TSPA is a versatile architecture and can be applied to various image recognition tasks. Here, we apply it to the task of license plate recognition. In particular, we utilize YOLOv3-tiny [8] to train the license plate detection model (stage 1) and the character recognition model (stage 2). Then, we transfer these two trained models to the edge computing device. Our experimental results demonstrate that the propose TSPA can achieve real-time performance (for the task of license plate recognition) with edge computing.

The rest of this paper is organized as follows. In Section II, we introduce our proposed framework and its application to license plate recognition. In Section III, we describe the datasets used in our experiments and the experimental results. Finally, we provide concluding remarks in Section IV.

II. Proposed Method

Figure 1 shows the architecture of our proposed method. The details are explained as follows.

A. Edge Computing Device

In this research work, we choose Xilinx Zynq UltraScale+ MPSoC ZU5EV as the edge computing device [9]. It has the advantages of high computing speed, low capital cost, and low energy consumption. The key specifications of Zynq UltraScale+ MPSoC ZU5EV are shown in Table. I.

979-8-3503-0195-3/23/$31.00 ©2023 IEEE
DOI 10.1109/IS3C57901.2023.00014

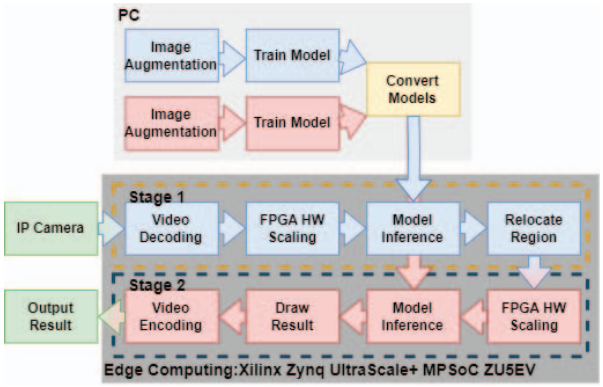

Fig. 1. The architecture of our proposed method.

TABLE I
KEY SPECIFICATIONS OF XILINX ZYNQ ULTRASCALE+ MPSOC ZU5EV

	Zynq UltraScale+ MPSoC ZU5EV
Price	$250
Machine Learning Throughput	1.36 TOPS
Watts/stream	5
Video encode and video decode	4K@60FPS

At present, the common deep learning frameworks include TensorFlow, PyTorch, and Caffe. The models trained by these frameworks are different from the data formats that edge computing devices can read. And some frameworks store model content by storing the model structure and model weight separately, so it cannot be directly used on edge computing devices. Therefore, the following three steps are required to successfully convert the model.

The first step is frozen graph. It is because the original model is stored by separating the model structure and weights, so the model structure and weights must be combined through frozen graph. The second step is quantization, which is to adjust the weight parameters that may have been in float32 or other formats to weight parameters in int8 format. The reason is that edge computing devices need to take into account the model size and computing speed, so they will optimize as much as possible. The last step is model compile. That is to recompile the model converted after the previous steps into a format that can be read by the edge computing device, so that the data processing unit can be successfully accessed and the subsequent inference can be performed.

B. Two-Stage Pipelined Algorithm

In stage 1, we firstly perform video decoding to convert the video stream from NV12 format to BGR format. Then, the decoded images are down-scaled through the FPGA hardware. The down-scaled images are fed into the first model for inference, which produce a region of interest (ROI) for usage in the subsequent stages.

In stage 2, the cropped ROI is up-scaled through the FPGA hardware. Then, it is fed into the second model for inference.

Fig. 2. Examples of AOLP dataset.

The inference results of the second model are drawn on the ROI. Finally, we convert the image frames from BGR format back to NV12 format to generate the output video.

Our proposed TSPA is a versatile architecture suitable for many real-world recognition tasks, such as license plate recognition (license plate detection + character recognition), face recognition (pedestrian detection + face recognition), text recognition (text area detection + text recognition), and dead leaf detection (plant detection + dead leaf detection). In the TSPA, the input of the second stage is a cropped ROI from the original image. Therefore, the inference model of the second stage only needs to process images of much smaller size. This can effectively improve the accuracy of the recognition task without sacrificing the processing speed.

C. Apply TSPA to License Plate Recognition

In this research work, we apply the proposed TSPA to the task of license plate recognition. The detailed settings of our application scenario are as follows. The resolution of the input video from an IP camera is 1920×1080. The input video frame is down-sized to 416×416 via FPGA hardware scaling. Then, we utilize YOLOv3-tiny [8] as the stage 1 model to detect license plates in the down-scaled images. Once a license plate is detected, it is cropped and sent to the second stage. During the second stage, the cropped license plate is firstly up-scaled to 512×224. This resolution is designed to conform with the aspect ratio of the license plates in Taiwan. We also perform zero padding so that it becomes 512×512 and feed it to the stage 2 model. We train another YOLOv3-tiny as the stage 2 model to recognize characters in a license plate region. The results of character recognition are drawn on the original images. Finally, the resulting images will be converted to the output video. Our goal is to demonstrate the feasibility of applying the proposed TSPA to the task of license plate recognition. Furthermore, the whole process can meet the real-time performance requirement (i.e., 30 fps).

III. EXPERIMENTAL RESULTS

In this study, we use the AOLP dataset [10] as the training set and self-collected videos as the testing set. The AOLP dataset have 2,049 license plate images. Figure 2 shows some

TABLE II
DETAILS OF THE SELF-COLLECTED TESTING VIDEOS

	Viewing Angle	Num. of frames	Num. of cars	# of totally appear	# of not appear	# of not fully appear
Video 1	45°	1326	7	681	541	104
Video 2	45°	1372	7	1130	136	106
Video 3	45°	1286	7	814	396	76
Video 4	Frontal	2047	15	1325	578	144
Video 5	Frontal	2220	18	1744	343	133
Overall	45°/Frontal	8251	54	5694	1994	563

TABLE III
EXPERIMENTAL RESULTS OF LICENSE PLATE DETECTION

	TP	TN	FP	FN	Accuracy
Video 1	446	541	0	235	0.8076
Video 2	1000	135	1	130	0.8965
Video 3	549	391	5	265	0.7768
Video 4	980	535	43	345	0.7961
Video 5	1531	338	5	213	0.8955
Overall	4506	1940	54	1188	0.8384

TABLE IV
EXPERIMENTAL RESULTS OF CHARACTER RECOGNITION

	Correct	Wrong	Precision
Video 1	238	208	0.5336
Video 2	771	229	0.7710
Video 3	353	196	0.6429
Video 4	787	193	0.8030
Video 5	1178	353	0.7694
Overall	3327	1179	0.7383

exemplar images from the AOLP dataset. On the other hand, our dataset contains six self-collected videos, ranging in length from 44 to 74 seconds. These videos have resolution of 1920×1080 and are stored in NV12 format. To mimicry the real situations, we capture these videos from two viewing angles, i.e., frontal view and 45 degree angle view. In Figure 3, the left column shows the frontal view images and right column shows the 45-degree-view images. When we capture the video of a moving vehicle, it moves toward to the camera and then leaves. Thus, its license plate only appears in some video frames. For the other video frames, the license plate either partially appears or does not appear at all. We only choose the video frames in which the license plates fully appear (positive samples) or not appear (negative samples) for testing our proposed method. Table II summarizes the details of our self-collected videos.

Notice that most of the vehicles in AOLP dataset are sedans and the vehicles in the self-collected videos are trucks or trailers. In Taiwan, the license plates of sedans have white background with black characters, while the license plates of trucks have green background with white characters. This poses a challenge for the recognition task because they have opposite contrasts between background and foreground colors. In order to deal with this issue, we propose to perform color inversion on the AOLP dataset. Given an images from the AOLP dataset, we firstly utilize histogram equalization to improve the image contrast. Then, each pixel value of the contrast-enhanced image is inverted as follows:

$$p' = p_{max} - p \tag{1}$$

where p denotes a pixel value, p_{max} denotes the maximum pixel value of this image, and p' denotes the inverted pixel value. This will convert the license plate region to dark background with white character as shown in Figure 4. After performing color inversion on the AOLP dataset, its license plate regions will have similar contrast as the license plate regions in our self-collected videos. This helps to train a more effective model for the target dataset. In fact, our training set includes the original images and the images after color inversion. Thus, we can not only mitigate the domain shift problem but also increase the number of training samples.

A. Evaluation of Stage 1 Model

For the task of license plate recognition, the stage 1 model is responsible for detecting license plate. We use manually labelled images from the AOLP dataset to train a YOLOv3-tiny [8] object detector. In particular, we have a training set of $4,372$ images and validation set of $1,100$ images. Then, we perform testing on our self-collected videos. The testing set contains six videos of totally $8,251$ frames. As mentioned earlier, license plates may partially appear in some of these video frames. We do not include the video frames with partially occluded license plates (563 frames) in this experiment. Therefore, we have $5,694$ positive samples (with license plate) and $1,994$ negative samples (without license plate) for evaluating the accuracy of license plate detection.

The detection accuracy is defined as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

TP : There is a license plate in the frame and the license plate is detected.

TN : There are no license plate in the frame and no license plate is detected.

FN : There is a license plate in the frame but no license plate is detected.

FP : There is no license plate in the frame but a license plate is detected.

The YOLOv3-tiny detector achieves true positive rate of $4506/5694 = 0.7914$, true negative rate of $1940/1994 = 0.9729$, false positive rate of $54/1994 = 0.0271$, false negative rate of $1188/5694 = 0.2086$, and detection accuracy of 0.8384.

The processing times of each step in stage 1 are as follows: video decoding takes 10.08 ms, image down-scaling takes 4.67 ms, model inference (i.e., license plate detection) takes 11.09 ms, and relocate region takes 0.93 ms.

B. Evaluation of Stage 2 Model

After finding the region of license plate, the stage 2 model will then perform character recognition. In Taiwan, the license plates contain both Capital English alphabets (without "I" and "O") and Arabic numerals (from 0 to 9). Again, we use manually labelled images from the AOLP dataset and internet to train a YOLOv3-tiny [8] model. We have a training set of $8,212$ images and validation set of $2,056$ images. Then, we perform testing on our self-collected videos. For the task of character recognition, one needs to correctly identify all the characters on a license plate. In other words, the recognition result is considered as wrong if one or more characters are not correctly identified. Therefore, we define the precision of character recognition as follows:

$$Precision = \frac{Corrects}{Corrects + Wrongs}$$

$Corrects$: All symbols are correctly recognized.

$Wrongs$: Not all symbols are correctly recognized.

The YOLOv3-tiny detector achieves the precision of 0.7383. The processing times of each step in stage 2 are as follows: image up-scaling takes 0.93 ms, model inference (i.e., character recognition) takes 6.68 ms, drawing the inference result takes 0.23 ms, and video encoding takes 48.67 ms.

Therefore, the total processing time is 82.35 ms and the resulting frame rate is about 12 fps. For some application scenarios (e.g., parking lot access control), the license plate recognition system only needs to generate license plate number rather than an output video. The processing time would reduce to 33.45 ms once we discard the steps of drawing the inference result and video encoding. This demonstrates that our proposed approach can achieve real-time performance (i.e., 30 fps) using edge computing.

IV. Conclusion

In this paper, we propose a versatile framework, called TSPA, for various recognition tasks. We apply TSPA to the task of license plate recognition. We also propose a color inversion method to handle the license plates of different types of vehicles, which have different background colors and character colors. We have conducted preliminary experiments using AOLP dataset as the training set and the self-collected videos as the testing set. Our proposed method can effectively detect license plate and recognize characters on a license plate. The experimental results demonstrate the feasibility of applying the TSPA to a specific task.

Fig. 3. Our self-collected dataset contains videos with two different viewing angles, (Left) frontal view; (Right) 45 degree angle view.

Fig. 4. Example of Color Inversion.

References

[1] F. Wang, M. Zhang, X. Wang, X. Ma, and J. Liu, "Deep learning for edge computing applications: A state-of-the-art survey," *IEEE Access*, vol. 8, pp. 58 322–58 336, 2020.

[2] S. Lee, K. Son, H. Kim, and J. Park, "Car plate recognition based on cnn using embedded system with gpu," in *2017 10th International Conference on Human System Interactions (HSI)*, 2017, pp. 239–241.

[3] S. T. H. Rizvi, D. Patti, T. Björklund, G. Cabodi, and G. Francini, "Deep classifiers-based license plate detection, localization and recognition on gpu-powered mobile platform." *Future Internet*, vol. 9, no. 4, p. 66, 2017.

[4] L. S. Fernandes, F. H. Silva, E. F. Ohata, A. Medeiros, A. V. L. Neto, Y. L. Nogueira, P. A. Rego, and P. P. R. Filho, "A robust automatic license plate recognition system for embedded devices," in *Brazilian Conference on Intelligent Systems (BRACIS)*, 2020, pp. 226–239.

[5] C.-J. Lin, C.-C. Chuang, and H.-Y. Lin, "Edge-ai-based real-time automated license plate recognition system," *Applied Sciences*, vol. 12, no. 3, p. 1445, 2022.

[6] D. M. F. Izidio, A. P. A. Ferreira, and E. N. S. Barros, "An embedded automatic license plate recognition system using deep learning." *Design Automation for Embedded Systems*, vol. 24, no. 1, pp. 23–43, 2020.

[7] H. Padmasiri, J. Shashirangana, D. Meedeniya, O. Rana, and C. Perera, "Automated license plate recognition for resource-constrained environments." *Sensors*, vol. 22, no. 4, p. 1434, 2022.

[8] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[9] "Zynq ultrascale+ mpsoc," *https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html*, 2023.

[10] G.-S. Hsu, J.-C. Chen, and Y.-Z. Chung, "Application-oriented license plate recognition," *IEEE transactions on vehicular technology*, vol. 62, no. 2, pp. 552–561, 2012.