Deep Learning with Accelerated Execution: A Real-Time License Plate Localisation System

Jimmy Ma* and Zoran Salcic**
Department of Electrical, Computer and Software Engineering
University of Auckland
New Zealand
*ma417704684@gmail.com and **z.salcic@auckland.ac.nz

2021 IEEE 19th International Conference on Industrial Informatics (INDIN) | 978-1-7281-4395-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/INDIN45523.2021.9557474

***Abstract*—In this paper, a real-time, real-life, novel license plate localisation (LPL) based on deep learning (DL) and accelerated by field-programmable gate array (FPGA), and Open Visual Inference and Neural network Optimization (OpenVINO) toolkit, is proposed and prototyped. The solution was tested against two popular international research databases and achieves state-of-the-art results, proving the viability of FPGA in real-life latency-oriented application. Using novel asynchronized DL inference that prepares next result while current inference is ongoing, the system increases computational efficiency without buffering frames, allowing for reduced latency. Comparisons show that the proposed LPL system has lower latency and better performance per watt than other related solutions.**

***Index Terms*—License Plate Localisation, FPGA Acceleration, Deep Learning, Real-time**

I. Introduction

Deep learning (DL) is one of the most promising techniques that significantly enhances performance in the computer vision applications such as object detection, object recognition and object segmentation.

Multi-layer perception (MLP) and convolutional neural network (CNN), two main categories of DL architectures applied in computer vision applications, are capable of enhancing robustness and generalizability. Recently, the CNN-based architectures are getting more attention than MLP. However, to achieve high accuracy and precision, the state-of-the-art CNN-based architectures normally contain millions of parameters which lead to a huge computational cost, and result in a longer inference time. This is an obstacle to adoption of CNN-based approaches in real-time applications, especially on edge devices [1].

In order to reduce latency and enhance throughput, algorithm (software) optimization and hardware acceleration are two dominant approaches explored in research and industry. Software optimizations are focused on the DL process, such as network pruning, quantization for low precision inference [2] and some novel DL model structures, e.g., depth-wise separable convolution [3], and the Single Shot MultiBox Detector (SSD) architecture [4]. However, hardware acceleration approaches are emerging recently, shortening the inference time while without significant sacrifice in terms of the accuracy.

Intel's Open Visual Inference and Neural network Optimization (OpenVINO) framework provides several out-of-the-box techniques for acceleration of DL, supporting a variety of execution platforms including CPU, GPU, field-programmable gate array (FPGA), vision processing unit (VPU) or heterogeneous computing (HETERO) [5]. The use of field-programmable gate arrays (FPGA) is a promising approach for using digital hardware to speed-up DL inference by employing significant parallelism, well-suited for convolution operations which are critical parts of CNNs [6]. Moreover, an important advantage of FPGA over GPUs is the low latency operation and the direct supply/delivery of input/output data, without using CPU, making it suitable for real-time object detection applications such as automatic license plate recognition (ALPR) and many applications from manufacturing and industrial automation domain.

ALPR plays a crucial role in many intelligent transportation systems or smart parking solutions. Specifically, license plate localisation (LPL) and license plate recognition (LPR) are two main components of an ALPR systems. This paper proposes a novel LPL based on the work presented in [7], which investigated the feasibility of DL with accelerated execution using OpenVINO with FPGA acceleration. In this paper, we investigate the feasibility of DL software and/or acceleration hardware by employing the OpenVINO framework and an FPGA card added to a standard CPU in a LPL real-time system focusing on latency, speed, and power consumption.

The main contributions of this paper are:

1) An accurate and robust LPL architecture based on a state-of-the-art CNN model leveraging the OpenVINO techniques accompanied with FPGA acceleration.
2) A pipelined video capture and inference approach, which increases the speed and decreases latency by allowing the LPL system to capture the next frame and execute inference while the current frame is being extrapolated.
3) LPL system testing and validation on a real-world real-time China public road showing 85.59% accuracy of the license plate region localisation over 111 vehicles that pass beneath the video capture setup [8].

The rest of the paper is structured as follows. Section II contains relevant background information. Section III describes the details of a real-time LPL system and Section IV compares its performance with other two state-of-the-art proposals. Section V offers a conclusion and potentials future works.

II. Background

In this section, we briefly introduce the case for using FPGAs to accelerate DL, and provide a basic overview of concepts of the OpenVINO. We also discuss the object detection architectures, especially SSD which is used in our LPL system. We also briefly introduce our LPL system architecture.

A. FPGA acceleration for DL

FPGAs have proven to be suitable for specialised/customised computation. For instance, FPGA has better performance per watt than GPUs on execution of subroutines and repetitive computations, which is important to DL, such as sliding-windows computation [9]. They also offer flexibility through reconfigurability [10]. However, the traditional approaches to develop FPGA-based applications by using hardware description language (e.g. VHDL, Verilog), requires specialist hardware design knowledge. Only recently, OpenCL started to support development for FPGAs, providing a standard interface for open parallel computing, thus enabling the user to develop FPGA-based applications using software-level specification and standard programming languages like C and C++.

B. OpenVINO Toolkit

OpenVINO is an open-source toolkit that focuses on accelerating DL execution pipeline in computer vision. Being supported by Intel, OpenVINO provides plug-and-play optimization for different DL frameworks that are widely applied in industry and research, such as Caffe, TensorFlow, MXNet and Kaldi. Also, OpenVINO enables heterogeneous computing for DL by supporting mix of different hardware targets, including CPU, GPU, FPGA and VPU.

In the deployment workflow of Open VINO Model Optimizer (MO) and Inference Engine (IE) are two key components. MO is a cross-platform command-line tool which optimizes trained DL models and converts them into a proprietary format; IE provides a unified API to cover different targets from a range of Intel's platforms including CPU, GPU, FPGA and VPU. Also, some software-level optimizations such as Low-Precision 8-bit Integer Inference are offered in IE. More detail can be found in [7] and [5].

C. MobileNet

The CNN is a standard application of DL, inspired by way of human mind learning [11], and proved very successful at solving problems such as object detection. However, being predominantly comprised of convolution layers, prior CNNs such as GoogleNet and SqueezeNet have millions of parameters resulting in huge computation time, and as such deemed unsuitable for real-time systems.

MobileNet targets embedded systems and mobile devices by reducing the number of layers and leveraging separable convolution [3], contains a depth-wise convolution followed by a point-wise convolution and has fewer parameters than a non-separated convolution, reducing inference time and making the model less prone to over-fitting [12]. A depth-wise convolution performs filtering over multiple channels using the same amount of kernels to produce separate outputs before combining/stacking them [12]. A point-wise convolution applies a 1 x 1 spatial filter across the outputs of depth-wise convolution and fuses them using linear combination [3], [12].

As a two steps process of filtering and combining, compared with a standard convolution, the separable convolution [7] results in a reduction of the computation of $\frac{1}{N} + \frac{1}{D^2}$, where N is the number of output channels and D is the kernel size. In our LPL system, we choose a MobileNet model which uses a kernel of 3×3 and its computational cost is about nine times smaller than that of standard convolutions.

D. SSD

A number of object detection systems based on CNN has been proposed in recent years. Unlike conventional approaches like edge detection and histogram of oriented gradient, CNN-based architectures for object detection are more robust and generalisable due to high-dimensional feature extraction. Prior architectures like R-CNN, Fast R-CNN, Faster R-CNN [13] (FR-CNN), and R-FCN [14] have achieved high accuracy while combining image classification and object localisation. However, these architectures are too computationally intensive for real-time application even with high-performance hardware and not well-suited for our LPL system.

More recent architectures like SSD [4], and You Only Look Once (YOLO) [15] convert object detection to a regression problem, to output object location and class probabilities in one evaluation by introducing bounding boxes. These methods significantly reduce the processing time of each video frame. In particular, by applying different bounding boxes on different sizes of the feature map, the SSD approach is capable of handling the different sizes of the object. Moreover, in our LPL system, we can set up the scale of default boxes in the training stage as per aspect ratios of a license plate, to achieve better performance. In our proposal, different than a VGG-16 network in [4], we select a MobileNet model as the cornerstone of SSD approach, to enable the architecture to run in real-time.

E. LPL

LPL is the most important component of an ALPR solution. LPL system detects that an image includes one or multiple license plates and then finds where they are in the image. In conventional approaches, e.g. [16], [17] proposed an LPL system using the sliding window technique and variations; [16], [18] provide modifications of edge detection analysis; mathematical morphology was used in [19]; and histograms of oriented gradients are selected in [17]. Due to human-constructed features, it is difficult for these traditional methods to get high accuracy in a complex environment such as overexposure and low brightness. On the other hand, DL-based methods tend to gain more reliable and robust performance, extracting abstract features automatically. In the early works, sliding window in addition to CNN classifier was often used. First, some rectangles in different scale and size are selected.

Then, they slide from the top left corner of the target image to the right bottom corner. The classification of the region within the rectangle is done after every slide. Finally, the location of the rectangle being considered a license plate by the CNN is recorded.

In subsequent works, region-based methods are adopted. Utilizing region proposal network (RPN), higher accuracy than before is achieved in LPL systems via three-step mechanism:(1) A trained RPN proposes candidate regions, (2) a CNN extracts features from each region and (3) the CNN is used to classify regions. Region-based proposals like R-CNN, Fast R-CNN and FR-CNN [13], have significant performance in terms of accuracy, but are very computationally intensive even when using high-end hardware and impractical to deploy on low power devices. LPL using region based methods are shown in [20],claiming accuracy of 99% and a recall of 81% but needing average 60ms to process one image while using a Core i7 CPU with an NVIDIA GTX980 GPU.

SSD [4], and YOLO [15] architectures are the state-of-the-art relevant to object detection, generating object locations and object class probabilities in one evaluation. These architectures are suitable for low power devices such as embedded systems. [21] proposes an LPL system utilizing the SSD architecture; they implement a real-time solution on Raspberry Pi 3 with an Intel Neural Compute Stick 2 (NCS2) and achieve 13 FPS for real-time license plate capture.

F. Our proposal for LPL

We propose a new real-time LPL (RTLPL) DL-based system to demonstrate the feasibility of FPGA acceleration, and show how to optimise the performance for a real-time, low power and cost-sensitive object detection application. The key points of our LPL system are that it is trained using MobileNet-SSD model on the BIT-Vehicle dataset for LPL, and it uses Terasic OpenVINO starter kit set as a hardware platform. The platform includes a low power Intel Celeron N3350 processor and an Intel Cyclone V FPGA. OpenCV is used to pre- and post-process images. Our proposal resulted in a better trade-off between performance and power consumption compared with [19], [21]. More details on comparison are provided in Section IV-C.

III. RTLPL - Real-time license plate localisation system

The system goal is rather challenging because license plates often appear as small objects within a big complex image, and consequently are difficult to detect accurately. It all needs to be done with a very low latency, for instance, LPL can be utilized at the entrance of a parking or at an intersection to read numbers of license plates. The accuracy of localisation along with real-time requirement make this a perfect goal to show capabilities of FPGA-enhanced LPL system.

Figure 1 depicts the design flow. It involves the following steps:

- We choose MobileNet-SSD as the target CNN architecture and then TensorFlow 1.13.0 to train it on the BIT-Vehicle [22] dataset.
- We utilize the MO to convert a fixed graph produced by the TensorFlow and then FP16 (16-bit floating point) IR files.
- We use OpenCV 3.4.2 to capture and resize image from a webcam and then upload to FPGA + CPU (HETERO Plugin) through C++ OpenVINO 2019 R1 APIs.
- We render results onto the original frame and display it on screen using OpenCV.

A. CNN Configuration and Training

We start with a Python program to setup initial parameters for a MobileNet-SSD model, and then train it on BIT-Vehicle dataset [22] by using TensorFlow APIs. BIT-Vehicle dataset archives 9,850 vehicle images from a China motorway. There are images with 1600x1200 and 1920x1080 pixels captured from two cameras at different time and place. The images contain changes in the illumination condition, the plate sizes, colour of license plate, and the car speeds. Exemplars of an image is shown in Figure 2. There may be one or two vehicles in one image and location of vehicles and their license plates are pre-annotated. Due to fixed location of camera, all vehicles are captured from the overhead and only include front sides. The database also provides the width, the height, the bounding boxes (i.e. ground true boxes) of license plate of each image. A bounding box contains the coordinates, width and height of a license plate. Each image can have multiple license plates, depending on the number of vehicles in the camera view.

1) MobileNet-SSD: In our system we choose MobileNet-SSD to execute DL inference in the LPL system. MobileNet [3] aims to execute DL inference on low-power devices by utilizing depth-wise convolutions and point-wise convolutions. On the other hand, the SSD detector [4] is an add-on CNN architecture which focuses on increasing the efficiency of object detection, retrieving multiple objects with multiple sizes in one inference. More details can be found in [7].

2) Initial configuration parameters: The most important parameters for SSD architecture are default boxes which are basically predicted scales for target objects. In our application, license plate is the only target we aim to detect so that we only need to set default boxes to fit scales of license plates. To mine suitable scales for default boxes, we refer to the k-means approach which is used in the YOLO v2 DL model. The main idea of this algorithm is substituting Intersection over Union (IoU) value for the Euclidean Distance and then clustering k license plate clusters. Moreover, the input sizes of our MobileNet-SSD are 256x256 and 300x300 pixels which are different from size of dataset image, 1920x1080 and 1600x1200, respectively. Thus, we need to convert the coordinates and size for license plates in the training set before clustering. The steps of algorithm are as follows:

- Randomly choose k ground true boxes in the training set.
- Calculate IoU with these k boxes for every box of the training set, and then assign every box to one box which

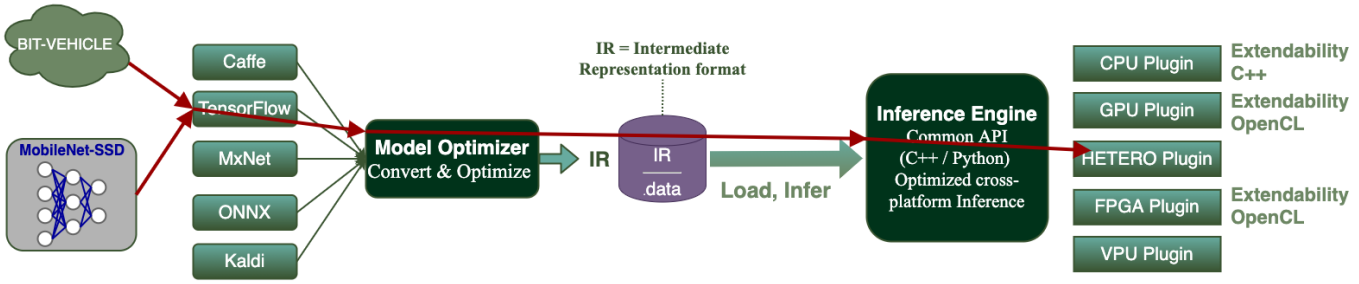


Fig. 1: A design flow starts from MobileNet-SSD to HETRO Plugin.

(a) (b) (c) (d)

Fig. 2: Different conditions of license plates in images from the BIT-Vehicle dataset. (a) Rotated, underexposed plate, (b) Blurry plate, (c) Bended plate, (d) Shadow-covered plate.

has the largest IoU value. The k ground true boxes and their assigned boxes form k clusters.

- Calculate a mean box for every clusters to be new k boxes and then conduct the step 2 again.
- Repeat the last two steps until the k boxes do not change or number of iterations reached a threshold.

We set k = 10 by cross validation and run this algorithm, then we get result as shown in Figure 3, while x axis and y axis indicate the width and the height of license plates in the training set. Because we only need to detect license, it is unnecessary to use five scales like in [4] and we choose three scales a_r from the result, $a_r \in \{1.8, 2.0, 2.6\}$ (width/height). By reducing the number of default boxes we reduce the training time and inference time. Default box can be simply regarded as predicted scales of targets and is described in [4], [7].

Then we calculate scale of the default boxes for each feature map just like [4]:

$$s_k = s_{min} + \frac{s_{max} - s_{min}}{m - 1}(k - 1), k \in [1, m] \qquad (1)$$

where s_{min} is 0.2 and s_{max} is 0.9 and other feature maps which belong to SSD architecture in between are regularly spaced. We impose width $w_k^a = s_k\sqrt{a_r}$ and height $h_k^a = \frac{s_k}{\sqrt{a_r}}$ for each default box following the method of [4] where k is index of feature map and a is the index of a_r, and then we do an additional computation because license plates are tiny

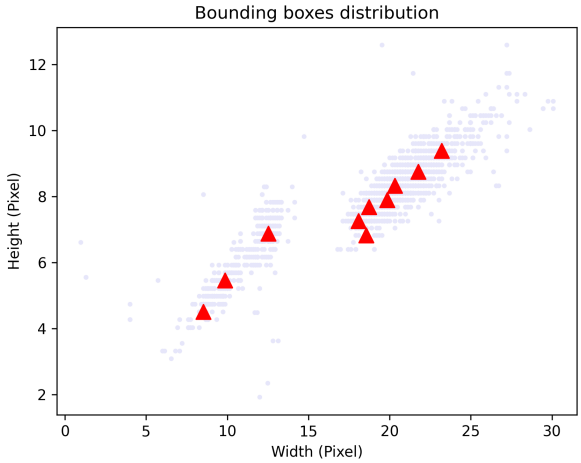


Fig. 3: Results of k-means algorithm to cluster bounding boxes of the training set

in common scene:

$$w_k^a = \frac{w_k^a}{10 \times \frac{s_{min}}{s_k}}, h_k^a = \frac{h_k^a}{10 \times \frac{s_{min}}{s_k}} \qquad (2)$$

By this conversion the model can detect smaller object as well as very large ones. Table I depict the width and height of default boxes computed for each feature map and the unit of default box size is the size of corresponding feature map.

Another important parameter is the learning rate. We ran

TABLE I: Default boxes for each layer

Feature map	Size of default boxes (Relative to size of feature maps)
Conv4_3	(0.027, 0.015) (0.028, 0.015) (0.032, 0.012)
Conv7	(0.078, 0.043) (0.082, 0.041) (0.094, 0.036)
Conv8_2	(0.153, 0.060) (0.163, 0.082) (0.185, 0.072)
Conv9_2	(0.257, 0.142) (0.269, 0.136) (0.307, 0.121)
Conv10_2	(0.388, 0.217) (0.407, 0.205) (0.464, 0.179)
Conv11_2	(0.545, 0.302) (0.572, 0.288) (0.653, 0.252)

65000 iterations in the training stage and start training with $4e^{(-4)}$ learning rate and then change it in regularly spaced steps after the every 1000 or 10000 iterations, as shown in table II. We also use as hyperparameters a batch size of 32,

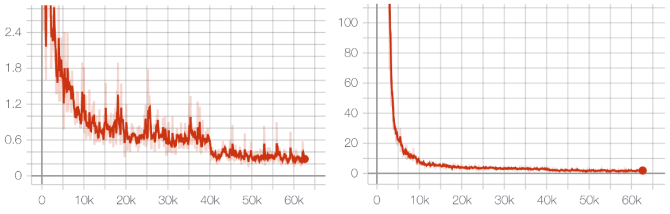
the Adam optimizer, and a momentum optimizer value of 0.95.

TABLE II: Learning rates in training stages

Iteration	Learning Rate
0-1000	0.0004
1000-10000	0.01
10000-40000	0.005
40000-55000	0.0005
55000-65000	0.00005

3) Training: To reduce training time, we train the model on the Baidu Brain Open AI platform [23] equipped with a NVIDIA Tesla V100 GPUs and Intel Xeon CPUs, Ubuntu 16.04 LTS operating system, and TensorFlow-GPU 1.12.0, Python 3.6, CUDA 9.0 and CUDNN 7.4.1.

To monitor the training, we write a script to evaluate localisation loss and classification loss along with the training. We use L1-smooth to calculate localisation loss and Cross Entropy to calculate classification loss, and we regard overall objective loss as the addition of weighted sum of the localisation loss and the classification loss as in [4]. Figure 4 (a) and (b) show the training process with epochs on x-axis and changes of localisation loss value and classification loss value, respectively. Figure 5 shows the overall loss value along the training.



(a) (b)

Fig. 4: Localization loss and Classification loss changes alone the training.

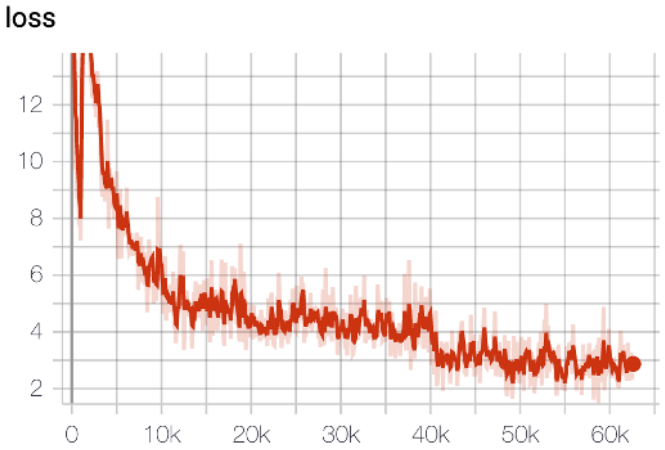


Fig. 5: Overall loss changes alone the training

B. System Configuration

Our LPL system prototype includes a webcam with 640x480 resolution and 30 FPS connected to Terasic OpenVINO starter kit set [24]. In addition, the system computer and FPGA board are powered by 5V and 12V power supply, respectively. The maximal power consumption for the computer is about 11 watts, and in stable operation it is 6 watts. The maximal power consumption for the Cyclone V FPGA is about 19 watts, and when running our implementation of CNN it was 12 watts.

C. RTLPL System Operation

Normally, a real-time application demands not only high frame rate but also low latency. There are many ways how this metric can be optimized. For example, the LPL system in [21] uses multiple threads video input to reduce frame capture time and increase frame rate; however, maintaining multiple threads is complicated. Also, accessing a critical section reduces performance of the application. Moreover, the latency will significantly increase when frames are cached because inference time is much longer than capture time.

We utilise the features provided by OpenVINO to implement asynchronized DL inference and achieve sufficiently fast and efficient performance without mutithreading. We create two inference requests, so that we can decode the next frame and feed it into one inference request while current frame is being extrapolated. This way, we change the time of processing one frame from:

$$OriginalTime = CaptureTime + InferenceTime \quad (3)$$

to

$$ReducedTime = Max(CaptureTime, InferenceTime) \quad (4)$$

More specifically, we create two inference requests from the inference plugin and each inference request can extrapolate DL model concurrently but share the same hardware implementation. The precondition for inferring multiple DL models in parallel is selecting asynchronized mode for each inference request. Figure 6 depicts the operation of our LPL system.

IV. RTLPL Experimental Evaluation

To evaluate performance of our RTLPL system, we define several metrics to measure performance and then compare it with other available solutions.

A. Performance Parameters

Frame processing time: As shown at top left corner of Figure 7, the RTLPL system displays a real-time FPS (frames per second) value on the screen. As the processing time of consecutive frames fluctuates, we adopt the method of moving average filter, where we calculate moving average for five consecutive frames.

Latency: A major advantage of FPGA based systems is low latency operation. Figure 8 illustrates our latency measurement process. We project a still image on the computer screen, with a millisecond resolution clock on top and time indicated

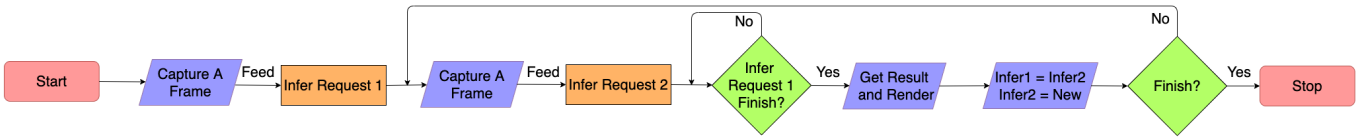


Fig. 6: A flow chart of our LPL system

Fig. 7: A screenshot of the display of RTLPL system in operation

as "t0". The output of our RTLPL system is shown on the same screen in a different window with the time on this window is denoted as "t1". The latency is calculated as the difference of t1 and t0. We average the latency over larger number of measurements (10 or more) to characterise system performance. **Mean Average Precision (mAP):** mAP

Fig. 8: A screenshot of latency measurement.

is another important metric, which is used to evaluate accuracy for a set of predictions. The calculation of mAP takes both Precision and Recall into account, and considers different thresholds of predictions. Specifically, for an object detection application, Precision is a measure of, "when your model guesses how often does it predict correctly?". In RTLPL, we calculate mAP using the PASCAL VOC 2012 method [25]; mAP takes the sum of the precisions over two classes (license plate and background) and run the test on 2,463 images of the BIT-Vehicle test dataset. **Matching Strategy:** We follow the matching strategy used in [7]. Specifically, we first verify if the predicted class is equal to ground truth and then go ahead to evaluate Intersection over Union (IoU) value whether it is larger than 0.5.

B. System Performance

Utilizing the above described methods we measure speed (FPS), latency and mAP for different DL models and different system implementation platforms. Table III shows results of the experiments. We can see that on an average the FPGA-CPU platform does almost two times better in terms of latency albeit outperforms CPU by only 8.5% in terms of FPS value with the 300x300 MobileNet-SSD DL model. Meanwhile, FPGA-CPU achieves a slightly lower mAP than CPU on the BIT-Vehicle test dataset. We also generated a I8 (8-bit integer) precision model by using the Calibration Tool, and 300x300 I8 model achieves an improved FPS and latency compared with normal FP32 precision although it also has a moderately decreased mAP value.

On the other hand, the 256x256 version of MobileNet-SSD achieves the shortest latency of 152ms and the highest 22.56 FPS with FPGA acceleration, and outperforms the sole CPU by 55.6% and 8.1%, respectively. Notably, the RTLPL system achieves a low latency which is near to the case with FPGA only by using I8 precision model, albeit with a significant reduction of mAP.

The FPS values with FPGA acceleration and using only CPU are approximately similar because accessing a webcam is very time-consuming and causes a bottleneck in our RTLPL system. A major advantage of FPGAs is the low latency. For RTLPL, latency is more critical than FPS as license plate needs

TABLE III: Comparison of measurements for different configurations our RTLPL system

Resolution	Arch.	Hardware	FPS	Latency (ms)	mAP
640x480	MobileNet-SSD 300x300	CPU	18.33	288	**90.63%**
640x480	MobileNet-SSD 300x300 I8	CPU	18.75	250	80.52%
640x480	MobileNet-SSD 300x300	FPGA-CPU	**19.89**	**166**	90.57%
640x480	MobileNet-SSD 256x256	CPU	20.86	342	**90.75%**
640x480	MobileNet-SSD 256x256 I8	CPU	21.17	185	77.72%
640x480	MobileNet-SSD 256x256	FPGA-CPU	**22.56**	**152**	90.72%

to be detected once it appear in specific scenario such as smart parking system.

Also, compared with [26], we uplift the mAP by setup to detect only license plates instead of including vehicles. The RTLPL system achieves over 90% mAP using both 256x256 and 300x300 DL models with the IoU threshold of 0.5. We believe it can achieve a higher mAP if using a bigger DL model and we plan to verify it in future works.

C. Comparison to other Solutions

Finally, in this section, we compare the performance of the RTLPL with approaches presented in [19] and [21]. [21] also build a LPL system based on DL and low-power devices which are a Raspberry Pi 3 and Intel Neural Compute Stick 2(NCS2). They propose an optimized SSD architecture applied on the LPL system; on measurement side, they give an average process time on NTUA medialab data set [27] and run on a PC equipped with Intel Core i7-2620M CPU (2.7 GHz) and 8 GiB of RAM and provide a FPS value when running on their Raspberry Pi 3 and NCNS2. [19] propose a LPL algorithm based on morphological operations which belong to a conventional computer vision technique. They also conduct some optics enhancements for the algorithm such as noise removal and Tophat transform.

NTUA dataset contains 571 images of vehicles and license plates in Greece, which are different from BIT-Vehicle dataset on which we trained our model. For example, colours are blue and yellow in Chinese license plates but Greek license plates are white; characters are alphanumeric in Greek license plates while Chinese license plates contain Chinese characters; scales (dimensions) of license plates are different in two countries; the most crucial is that all vehicles are forward to camera in BIT-Vehicle but NTUA contains various poses of vehicles. Therefore, it is trivial to compare mAPs of the RTLPL system and [21] and [19], and we only conduct comparison for frame rates on NTUA and FPS in the real-life video.

Table IV shows the result for average processing time on NTUA dataset for our RTLPL system, [21] and [19]. [21], having 20ms processing time per image, uses 8GB RAM and Intel i7-2620M which scores 1265 in multi-core benchmark with 35W Thermal Design Power (TDP). [19] requires the same time during the test, also utilizes an Intel 2.7 GHz i7 processor (not given precise model) and 8GB RAM. On the other hand, our RTLPL system achieves at best 51ms with the Intel Celeron 1.1GHz N3350 with only 6W TDP. Although FPGA has extra power consumption of 12W in this case, the total consumption is still lower than [21] and [19]. Latency is not provided in [21] and [19], but we believe the latency of our RTLPL is better than [21] because they utilize multithreaded architecture to capture frames, which means frames are cached before being processed.

Moreover, our RTLPL system achieves 62ms (256x256) and 71ms (300x300) average processing time while reference [21] and [19] are only three times faster than ours but with 6 times larger TDP and much more powerful processors. From this we can conclude our RTLPL system has a better performance per watt compared to the competitors.

In terms of FPS, Table V provides the result from the reference [21] and [19] when running in inference mode and compares them with our system. [21] use the same resolution as ours with lower values of FPS and TDP, while ours have 20.86 FPS and 21.56 FPS with FPGA; our RTLPL system in the case of only using CPU still has better performance per watt than [21] and enhances the latency with FPGA acceleration.

In contrast, [19] achieves high FPS while only having 3.7 W TDP. However, its resolution in number of pixels in image is four times lower than ours. We believe such a low resolution output cannot satisfy real-life applications if this output is applied in a user interface, such as using that output for displaying customers license plate in a smart parking system. A hyperlink to a video showing our RTLPL system operating in real-life and real-time environment on a Chinese public road can be found at [8].

V. Conclusion

In real-time systems, FPGAs have proven to be a well-suited technology to accelerate DL inference pipeline. Together with tool chains, such as Intel OpenVINO, they offer a series of optimization solutions for vision-oriented DL applications. In this work, we proposed a RTLPL system to demonstrate the feasibility and viability of FPGA acceleration in real-time application. Our system selected the MobileNet-SSD architecture which was trained on the Baidu Brain AI Platform using the BIT-Vehicle dataset. Testing achieved 90.75% and 90.63% correct detection accuracy over the same dataset, with 300x300 and 256x256 model sizes, respectively.

Using two OpenVINO inference requests to do inference simultaneously, the next frame is captured and then being extrapolated while waiting the current inference result, where

TABLE IV: Comparison of LPL algorithms on the NTUA Medialab LPR database

Description of system	Processing time per image (ms)	TDP (W)	Detection accuracy (different dataset)
Ours (300x300)	57	18	90.57
Ours (300x300,I8)	71	6	80.52
Ours (256x256)	51	18	90.72
Ours (256x256,I8)	62	6	77.72
Deeplearning-based [21] (224x224)	20	35	98.4
Morphological operations [19]	20	≈35	98.45

TABLE V: Further comparison of LPL systems on the NTUA Medialab LPR database

Description of system	FPS	Resolution	Devices	TDP, W
Deeplearning-based [21] (224x224)	13	640x480	Raspberry Pi 3 + NCS2	4.7
Morphological operations [19]	18.84	320x240	Raspberry Pi 3	3.7
Ours (256x256)	20.86	640x480	Celeron N3350	6
Ours (256x256)	21.56	640x480	Celeron N3350 + Cyclone V FPGA	18

OpenVINO provides APIs to help us accomplish asynchronized DL inference. Meanwhile, by leveraging the power of FPGA, our LPL system achieved a two times lower latency (152ms) than a sole CPU, generating as output a real-time video stream (640x480), and achieved 21.56 FPS.

As future work, we plan to investigate the possibilities of tailoring hidden CNN layers for FPGA implementation. Moreover, an extension to license plate recognition is being planned, which would benefit from using FPGA and OpenVINO tool chain. Finally, integration of video-grabbing within the FPGA will be investigated to accelerate the whole processing of captured video-streams.

REFERENCES

[1] X. Wang, Y. Han, V. C. M. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, p. 869–904, 2020. [Online]. Available: http://dx.doi.org/10.1109/COMST.2020.2970550

[2] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *CoRR*, vol. abs/1510.00149, 2016.

[3] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *ArXiv*, vol. abs/1704.04861, 2017.

[4] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," *ArXiv*, vol. abs/1512.02325, 2016.

[5] Intel. (2020) Openvino toolkit. [Online]. Available: https://software.intel.com/en-us/openvino- toolkit

[6] G. Dinelli, G. Meoni, E. Rapuano, G. Benelli, and L. Fanucci, "An fpga-based hardware accelerator for cnns using on-chip memories only: Design and benchmarking with intel movidius neural compute stick," *International Journal of Reconfigurable Computing*, vol. 2019, 10 2019.

[7] Z. S. Jimmy Ma, "Deep learning with accelerated execution: Towards a real-time video analysis system (being review)," *Neural Computing and Applications*, 10 2020.

[8] ——. (2020) Our lpl system in real-life real-time environment. [Online]. Available: https://www.youtube.com/watch?v=Lj3sX0gMvPQ

[9] J. Fowers, G. Brown, P. Cooke, and G. Stitt, "A performance and energy comparison of fpgas, gpus, and multicores for sliding-window applications," in *FPGA 12*, 2012.

[10] G. Lacey, G. W. Taylor, and S. Areibi, "Deep learning on fpgas: Past, present, and future," *ArXiv*, vol. abs/1602.04283, 2016.

[11] D. Kurpiel, "Localização de placas veiculares em vídeo usando redes neurais convolucionais profundas," Ph.D. dissertation, BSc thesis, Universidade Tecnológica Federal Do Paraná, 2018.

[12] E. Bendersky. (2018) Depthwise separable convolutions for machine learning. [Online]. Available: https://eli.thegreenplace.net/2018/depthwise-separableconvoluions-for-machine-learning

[13] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, pp. 1137–1149, 2015.

[14] J. Dai, Y. Li, K. He, and J. Sun, "R-fcn: Object detection via region-based fully convolutional networks," in *NIPS*, 2016.

[15] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788, 2016.

[16] P. Tarábek, "A real-time license plate localization method based on vertical edge analysis," *2012 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pp. 149–154, 2012.

[17] R. F. de Carvalho Prates, G. C. Chávez, W. R. Schwartz, and D. Menotti, "Brazilian license plate detection using histogram of oriented gradients and sliding windows," *ArXiv*, vol. abs/1401.1990, 2014.

[18] B. Enyedi, L. Konyha, C. Szombathy, and K. Fazekas, "Strategies for fast license plate number localization," *Proceedings. Elmar-2004. 46th International Symposium on Electronics in Marine*, pp. 579–584, 2004.

[19] J. Yepez and S.-B. Ko, "Improved license plate localization algorithm based on morphological operations," *IET Intelligent Transport Systems*, vol. 12, pp. 542–549, 2018.

[20] A. Naimi, Y. Kessentini, and M. Hammami, "Multi-nation and multi-norm license plates detection in real traffic surveillance environment using deep learning," in *ICONIP*, 2016.

[21] J. B. Yepez, R. D. Castro-Zunti, and S.-B. Ko, "Deep learning-based embedded license plate localisation system," *Iet Intelligent Transport Systems*, vol. 13, pp. 1569–1578, 2019.

[22] B. I. of Technology. (2020) Bit-vehicle dataset. [Online]. Available: https://drive.google.com/file/d/1suu7qXFQVPYL4hPOX_fbqSn35VQI2iH2/view

[23] Baidu. (2020) Baidu brain. [Online]. Available: https://ai.baidu.com/

[24] Terasic. (2020) Developer kit for openvino™ toolkit. [Online]. Available: http://osks.terasic.com/

[25] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. (2020) The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. [Online]. Available: http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html

[26] chuanqi305. (2020) chuanqi305-mobilenet-ssd. [Online]. Available: https://github.com/chuanqi305/MobileNet-SSD

[27] G. National Technical University of Athens. (2020) Medialab lpr database. [Online]. Available: http://www.medialab.ntua.gr/research/LPRdatabase.html