IET Intelligent Transport Systems

Research Article

IET Journals
The Institution of Engineering and Technology

License plate segmentation and recognition system using deep learning and OpenVINO

ISSN 1751-956X
Received on 17th July 2019
Revised 8th November 2019
Accepted on 7th January 2020
E-First on 28th January 2020
doi: 10.1049/iet-its.2019.0481
www.ietdl.org

Riel D. Castro-Zunti[1], Juan Yépez[1], Seok-Bum Ko[1] ✉

[1]*Department of Electrical and Computer Engineering, University of Saskatchewan, Saskatoon, Canada*

✉ *E-mail: seokbum.ko@usask.ca*

Abstract: As Intelligent Transportation Systems applications increase in prevalence, Automatic License Plate Recognition solutions must be made continually faster and more accurate. The authors propose an embedded system for fast and accurate license plate segmentation and recognition using a modified single shot detector (SSD) with a feature extractor based on depthwise separable convolutions and linear bottlenecks. The feature extractor requires less parameters than the original SSD + VGG implementation, enabling fast inference. Tested on the Caltech Cars dataset, the proposed model achieves 96.46% segmentation and 96.23% recognition accuracy. Tested on the UCSD-Stills dataset, the proposed model achieves 99.79% segmentation and 99.79% recognition accuracy. The authors achieve a per-plate (resized to 300 × 300 px) processing time of 59 ms on an Intel Xeon CPU with 12 cores (2.60 GHz per core), 14 ms using the same CPU and OpenVINO (a neural network acceleration platform), and 66 ms using the proposed low-cost Raspberry Pi 3 and Intel Neural Compute Stick 2 with OpenVINO embedded system.

1 Introduction

Automatic License Plate Recognition (ALPR) systems are designed to capture license plates from vehicles and extract identifying information without direct intervention from a human overseer. ALPR is considered a subfield of Intelligent Transportation Systems [1], services designed to enrich drivers and their interactions on the road.

ALPR systems find use in tools for law enforcement: vehicle identification and tracking, issuing fines, restricting area access to certain types of vehicles, and so on. Multiple vehicles in excess of 80 km/h (i.e. highway conditions) may pass a single camera setup at any time. Thus, low processing time and high accuracy is essential to a successful ALPR system.

An ALPR system generally performs the following steps sequentially. First, an image is taken. Then, vehicles and/or license plates are detected. Finally, character regions from the license plate are extracted (segmented) and interpreted using an optical character recognition (OCR) method [2].

Oftentimes, license plates within an image are captured blurred, skewed, occluded, with insufficient contrast or brightness, or otherwise unideal. This makes subsequent segmentation and OCR difficult to perform quickly and accurately via purely algorithmic methods.

We examined whether a deep learning (DL)-based solution could provide fast and accurate License Plate Segmentation and OCR (LPSOCR). Using OpenVINO [3] – a framework released at Intel's 2018 AI DevCon – our final LPSOCR network can be implemented onto a variety of hardware including GPUs, CPUs, VPUs, FPGAs, or some combination thereof. Compatibility with a low-cost Raspberry Pi 3 and an Intel Neural Compute Stick 2 (NCS2) is demonstrated. Usage on a CPU with and without OpenVINO is also shown.

In this paper, we propose a DL architecture for LPSOCR using depthwise separable convolutions (DSCs) and residual linear bottlenecks. Section 2 presents basic information on DL and associated topics. Section 3 describes related research in LPSOCR, focusing on DL-based methods. Section 4 describes our architecture and training method. Section 5 details our results. Finally, Section 6 provides conclusions and opens future work.

2 Background

DL is an application of artificial intelligence with multiple prediction layers to enhance the accuracy of a learned model [4]. DL research is ongoing and cross-discipline, and has demonstrated state-of-the-art performance in various tasks and topics from computer vision to speech recognition [5]. The decline of the consumer cost of GPUs and the rise of big data from which a DL apparatus may learn has contributed to the success and continued refinement of DL architectures [5].

A popular DL architecture is the convolutional neural network (CNN), which is modelled after the human brain and attempts to learn in a similar way to that of a human [6]. CNNs have proved incredibly successful at solving problems in many domains – including computer vision – because they are able to extract and organise complex low-level features.

DL and CNNs typically require many layers and parameters to be successful [7], putting such research and applications within the realm of mid- or high-end GPUs. These GPUs are expensive to develop/manufacture, expensive to the average consumer, and consume lots of power.

Several successful CNN-based object detection systems have been developed. These include R-CNN, Fast R-CNN, Faster R-CNN [8] (FR-CNN), and R-FCN [9], which perform object detection via image classification. However, the deep nature of these architectures is impractical for real-time inference, even using a high-end GPU [10]. Subsequent architectures, like single shot detector (SSD) [11], significantly improved processing times over the models in [8, 9]. SSD and similar architectures did this by utilising regression to have bounding boxes and class probabilities predicted from an input image in a single evaluation [10].

In our previous work [10], we proposed a neural network architecture for license plate localisation able to localise a license plate region from an image. The system can accurately localise a plate that occupies anywhere from 10% to the entirety of the total image size. We modified the architecture proposed in [10] to recognise 36 classes (for each letter and all digits 0 through 9) as opposed to only one license plate class. Furthermore, we retrained the network using different datasets ([12, 13]) comprised of cropped license plate images (e.g. Fig. 1). Unlike [10], the neural network model we propose in this paper is parameter-optimised using the OpenVINO platform, thereby lessening processing time.

Fig. 1 *Labelled license plate images from the Caltech Cars [12] dataset*

3 Related research

License plate segmentation and recognition, and OCR itself, is well-documented in academic literature. However, the need for consistently faster, more accurate, and more robust segmentation and recognition specifically for ALPR systems motivates its continued research within ITS.

Traditional image processing and computer vision techniques for segmentation and/or recognition used in ALPR solutions are many and varied, and include the following: template matching [14, 15], connected component analysis [16, 17], blob analysis [18], mathematical morphology [15], and DSP-based transforms [19]. These methods, however, tend to fail across unideal plate conditions, including dirt and shadows.

For statistical machine learning-based LPSOCR, the OCR system in [20] is segmentation-free and instead uses a 'sweeping OCR'. Trained using synthetic character crop regions, an OCR linear support vector machine classifier is swept across a license plate to infer characters and their positions using hidden Markov models. The character sequence with the highest probability is obtained using the Viterbi algorithm. The method in [20] reports 99% accuracy across two private license plate image datasets, with a sweeping OCR processing time of ~1.6 s per image.

We focus our related research on DL-based LPSOCR platforms given the nature of our solution. A system for character-based license plate detection and segmentation-free recognition is proposed in [21]. Like [20], their method sweeps a classifier over a plate in sliding window fashion to extract sequential features. The 9-layer CNN (6 convolutional + 3 fully connected) is trained to classify 36 classes, one for each letter and number. They use principal component analysis on the concatenated outputs of the fourth convolutional and final fully connected layer, and the resultant 256d feature vector is analysed by a bi-directional long short-term memory (BLSTM) network with a softmax over 37 classes. The 37th class is for spacing between characters. Finally, they use CTC to decode the output of the BLSTM. Using an NVIDIA Tesla K40c GPU and a database of 1.38×10^5 characters, they achieved a recognition accuracy of 92.47% across the AOLP dataset (Taiwan). However, the authors admit their sliding window method is too slow for real-time applications.

Another sliding window technique is proposed in [22]. To compensate for the additional complexity of the sliding window, the authors use a parameter-reduced YOLO network trained on 36 classes (35 characters, combining '0' and 'O' into a single class, and a background class). They train using the AOLP dataset on an Nvidia GTX970 GPU with 4 GB memory, and an Intel i7 CPU with 16 GB of memory. Over the AOLP dataset, they achieve a character recognition accuracy of about 78%, with about 800 ms to 1 s per vehicle image.

The system described in [23] performs LPSOCR by first binarising the license plate using global- and edge-based thresholds to improve contrast, and then passing the image to a 7-layer CNN with Softmax regression for symbol classification. Their CNN is trained using 36 classes, for each letter (A–Z) and number (0–9). Using a Tesla K80 GPU with 24 GB of memory and/or a Tesla M40 GPU with 12 GB of memory, their method achieves 94.28% recognition accuracy on the Caltech Cars (Rear) 2 [12] dataset, and 94.46% on the NTUA Medialab LPR Database [24], but no execution time is given.

In [25], character segmentation was performed using pure computer vision. First, a greyscale license plate image has its contrast enhanced. Then, canny edge detection is applied. Thereafter, a contour hierarchy method is used to locate a curve joining all similar-intensity regions. Finally, a bounding box is created per-character, with geometric qualities used to minimise the area of the box. Recognition is performed using a 6-layer (4 convolutional and 2 fully connected) CNN model trained with 37 classes – all 36 alphanumeric characters, and a non-character class – with output realised using a Softmax classifier. Their method achieves a per-character recognition accuracy of 94.8% on the Caltech Cars [12] dataset, but no execution time is given.

To segment and read characters on Brazilian plates, Montazzolli and Jung [26] used a modified YOLO architecture where the first 11 layers (trained on the PASCAL VOC dataset [27]) are taken as-is and 4 convolution layers are added. They trained using a public Brazilian license plate database to enhance problem-specific non-linearity. They specify 35 classes, with all alphanumeric characters being their own class except for a combined '0' and 'O' class. Heuristics and per-character semantic modifications are applied on output such that the 7 highest-probability detections per-plate matches a LLLNNNN pattern (L denotes a letter and N a number). Their system achieves 63.18% per-plate, and 93% per-character, recognition accuracy. On a (high-end) Titan X GPU, their system's execution time is 2.2 ms, and on a (mid-end) GeForce GT 750M, the time is 20.1 ms.

A modified YOLO darknet framework to detect and recognise Brazilian license plates is described in [28]. Their training/testing process uses a self-collected 'real-world' urban-environment license plate dataset of 4500 images (30,000+ characters). For character segmentation, they use a 12-layer convolutional network modified from CR-NET. Segmentation instances with >7 detected regions have sufficiently close regions combined; otherwise, low-probability detections are discarded. Their character recognition process is split into two separate networks for alphabet and numeric characters. Training utilises image negation for augmentation. For alphabet recognition, they use their segmentation CNN architecture. For digit recognition, they use their segmentation architecture but with the first four layers removed. They also exploit frame redundancy when detecting from videos. They achieve a per-plate recognition accuracy of 78.33% over their dataset's test set, beating the closest commercial software by 8.33%. The system's per-character segmentation accuracy was 95.97% and took 1.65 ms per detection. The per-character recognition accuracy was 90.37% and took 1.65 ms per single-character classification. Their experiments were performed on an NVIDIA Titan XP GPU with 12 GB of RA – a hardware setup unpractical for edge-based systems. An updated version of their system was presented in [29], in which they trained using more datasets (a total of 9967 plate images). Although their recognition network was largely unmodified, they vastly increased the types of dataset augmentations, developed region- and layout-based heuristics, and trained their modified CR-NET using 35 classes (all letters and numbers except for a combined '0' and 'O' class). They achieve an average end-to-end recognition accuracy of 94.7% and

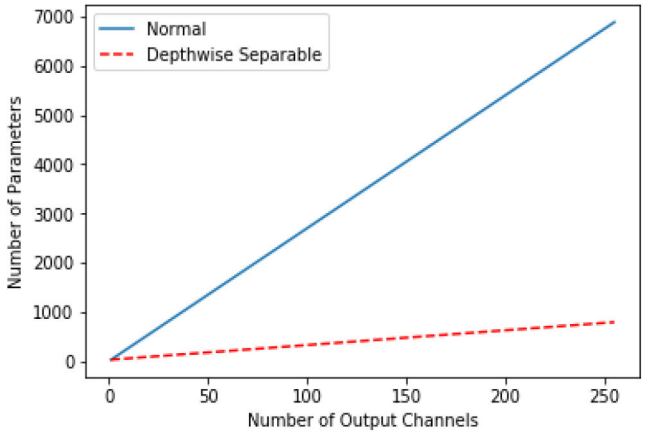

Fig. 2 *Number of parameters versus number of output channels using a 3 × 3 kernel and three input channels*

Table 1 Structure of bottleneck residual block

Input	Operator	Output
$h \times w \times k$	1 × 1 conv2d, ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3 × 3 dwise s = s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1 × 1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k$

Table 2 Feature extractor for license plate segmentation and recognition neural network

Input size	Operator	Repetition	Kernel
224 × 224 × 3	Conv2d	1	3 × 3
112 × 112 × 32	bottleneck	1	3 × 3
112 × 112 × 16	bottleneck	2	3 × 3
56 × 56 × 24	bottleneck	3	3 × 3
28 × 28 × 32	bottleneck	4	3 × 3
14 × 14 × 64	bottleneck	3	3 × 3
14 × 14 × 96	bottleneck	3	3 × 3
7 × 7 × 160	bottleneck	1	3 × 3
7 × 7 × 320	Conv2d	1	1 × 1
7 × 7 × 1280	Avgpool	1	7 × 7
1 × 1 × 1280	Conv2d	—	1 × 1

96.8%, respectively, with and without using their heuristics. Using the Titan XP GPU, their per-character recognition time is 1.99 ms.

The system in [30] recognises Bangla characters on license plates. Characters are segmented via image binarisation and contour detection, with 98% accuracy. OCR is performed with a 4-layer (2 convolutional and 2 fully connected) CNN with a Softmax classifier, and achieves 98% accuracy. A processing time of 110 ms for end-to-end (license plate detection and recognition) is provided, but it is not stated how much of that time is used only by LPSOCR. Neither the hardware specifications under which the experiments were conducted nor dataset information were provided.

How and Sahari [31] present a study of training an OCR neural network using computer characters in varying fonts and orientations from the Chars74K dataset [32], as opposed to using images of actual license plate characters. To recognise Malaysian license plates, a 6-layer (4 convolutional and 2 fully connected) CNN is trained with 35 classes (all numbers and letters excluding 'I' and 'O', and a background class). They achieved 95.89% per-character accuracy on a (private) dataset of real Malaysian license plate cropped characters. Hardware specifications and execution time are not provided.

In Section 5.2, comparisons are made against [23, 25, 29]. Those were selected because their solutions were trained and/or tested using the Caltech Cars [12] or UCSD-Stills [13] datasets, like our solution was.

4 Proposed solution

4.1 Neural network overview

Our proposed neural network solution uses a modified SSD architecture. SSD [11] was selected for three reasons. First, because of its capability for simultaneous segmentation and class-recognition of license plate characters. Second, because it can detect multiple detection boxes in a single forward pass. And third, because it operates faster than other methods like [8, 9].

The original SSD uses VGG-16 [7] as a feature extractor, which requires a high-end GPU for sufficiently fast operation. Our solution instead utilises a feature extractor that uses linear bottleneck depth-separable convolutions with residuals, similar to MobileNet [33]. DSCs lower parameters and thereby complexity, and their usage is well-suited to devices with low processing capabilities.

DSCs provide a faster method for convolution with negligible accuracy sacrifice. The number of parameters used in normal convolution is defined as

$$R \times S \times i \times o = \text{Parameter Count} \tag{1}$$

where R is the kernel width, S is the kernel height, i is the number of input channels, and o is the number of output channels. DSCs, however, are defined as

$$R \times S \times i + i \times o = \text{Parameter Count} \tag{2}$$

It can be noted that DSCs have incredibly fewer parameters than their normal counterparts as o gets large (see Fig. 2).

Any accuracy reduction is further mitigated via the use of linear bottleneck layers with ResNet-like [34] skip connections in which the input vector is added to the output of a block before processing the next block. A bottleneck layer is a low-dimensional representation of information relevant to a convolutional layer, and useful for classification and feature prediction [35]. 'Linear' simply refers to the fact that a non-linear activation function (such as ReLU) is not applied to the output of the block.

Unlike the original ResNet, in which skip connections are added between wide layers, we add them between narrow bottleneck layers: this idea gives the inverted residual block, the structure of which is found in Table 1. Note that the ReLU6 non-linear activation function is simply a normal ReLU but in which all inputs >6 are clipped at 6. Additionally, h refers to height, w to width, k to channels, and s to stride.

A detailed architecture for the feature extractor based on inverted residual blocks can be found in Table 2. A diagram of the object detection architecture can be found in Fig. 3.

4.2 Training process

For training, we used Tensorflow v. 1.8 on a computer running Ubuntu 18.04 LTS with a Tesla K40c GPU.

We trained/tested using license plate image crops from the Caltech Cars (Rear) 2 [12] dataset, comprised of one hundred and twenty six images of USA vehicles in Caltech parking lots. One hundred and twenty of these vehicles had Californian plates. Most images were taken in daylight conditions. Many exhibit proper exposure. There are few rotations or skews. Few plates have characters occluded to the point of being unrecognisable by a human. Several plates are blurry causing character regions to overlap, and others have shadows cutting into character regions. Added to our training set is the UCSD-Stills [13] dataset of license plate image crops with two hundred and ninety one images of vehicles with American license plates. Two hundred and eighty four of these vehicles had Californian plates. Plates were captured in daylight, but many were skewed, shadowed, or had unideal exposure. A few were blurry or partially occluded.

We trained with 36 classes, one for each alphanumeric character from A through Z and 0 through 9. From cropped license plates, character regions of interest were manually segmented and annotated using labelImg [36], which converted positional

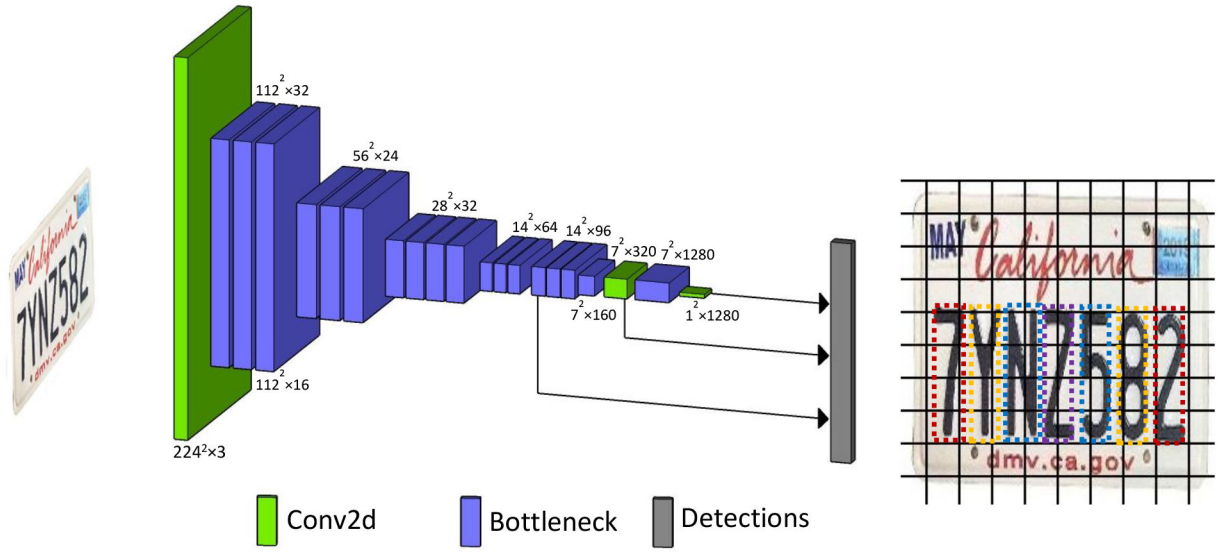

Fig. 3 *Architecture of our license plate segmentation and recognition model. License plate image from the Caltech Cars dataset [12]*

Fig. 4 *Varying transformations applied to a plate in the Caltech Cars [12] dataset*

(a) Normal, ***(b)*** Motion blurred, ***(c)*** Y-channel histogram equalisation, ***(d)*** HSV V-channel, ***(e)*** OpenCV BGR2GRAY, ***(f)*** Otsu's thresholding, ***(g)*** Inverse of Otsu's thresholding

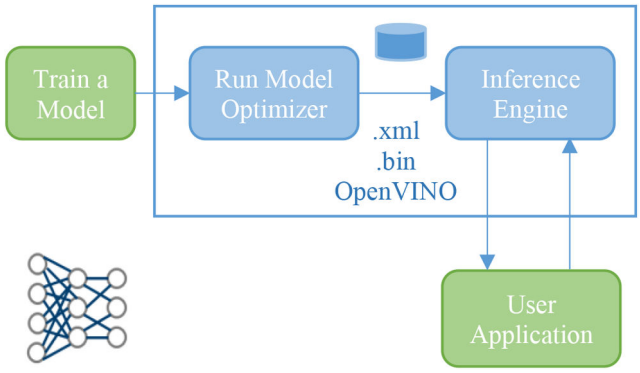

Fig. 5 *OpenVINO architecture*

bounding box and class data into an XML file processable by the DL training software.

Examples of labelled images can be found in Fig. 1.

Training a DL neural network model using stochastic gradient descent with backpropagation involves choosing components and hyperparameters. The training batch size used was 16. The initial learning rate was 0.004, lowered to 0.0005 after 355,000 steps. An L1 loss function was used during training. A dropout keep probability of 0.8 was used. The optimiser had a momentum of 0.9 with 0.9 decay and 1.0 epsilon. Input images were resized to a shape of 300×300 px. The model was trained to 562,000 iterations.

4.3 Dataset and augmentations

Per-plate data augmentation techniques were used to generate new training samples and thereby increase the amount of data. Our utilised transforms are as follows:

- Horizontal motion blurring, with blur amount dependant on the original license plate size (which varied per-image)
- Equalising the histogram of the Y channel in YUV colour space
- Taking the greyscale representation of the license plate via its V channel in HSV colour space
- Taking the greyscale representation of the license plate using the standard BGR2GRAY conversion in OpenCV
- Binarising the plate using Otsu's thresholding method
- Negating the binary image obtained from Otsu's method

Examples of these transforms are depicted in Fig. 4.

Transforms were selected to maximise class variation in texture (e.g. blur and histogram equalisation) and intensity (e.g. thresholding and greyscale conversion). This may allow our network to better distinguish actual features from noise, and thus we create a more robust solution.

Augmentations across both datasets brought our dataset to 2845 images for the learning process. Transformed and a random 80% of the original image-label data pairs are taken for training.

4.4 Optimised model and creation of an intermediate representation

The model obtained after training is not optimised for low-power performance. For fast edge inference on low-computation devices, we utilised OpenVINO [3]. OpenVINO performs static model analysis and redesigns a DL model for optimal execution on a target device, thereby easing the transition between training and deployment. As shown in Fig. 5, OpenVINO consists of two parts: the model optimiser (MO), and the inference engine (IE). OpenVINO creates files for an intermediate representation (IR)

using the MO, and the input to the MO is the network model trained using Tensorflow. The output of the MO is a model optimised for execution on a specific Intel CPU, GPU, VPU, FPGA, or a combination thereof. The MO optimises the model via the following mechanisms:

- Pruning extraneous model components that are required at the time of training, but not at the time of inference.
- Fusing operations. Some multiple operations can be combined into a single operation, and the MO detects such operations and fuses them.
- Bit-width reduction of weights (from 32- to 16-bit floating point) for compatibility with NCS2.

The result of the optimisation process is an IR model comprised of two files:

- model.xml, containing the network architecture.
- model.bin, containing the weights and biases.

Although the IR model is hardware agnostic and depends only on the neural network architecture, OpenVINO optimises the running of the model on specific hardware through the IE plugin. The IE plugin is available for all modern Intel hardware including GPUs, CPUs, VPUs, and FPGAs.

To accelerate the classification process, the trained model is deployed on an NCS2. Inference is performed using the IE, which requires as input the IR and the data to be analysed. The IE outputs a probability-based classification. The IE is a C++ library with a set of classes for inference from image data. The C++ library provides an API to read the IR, set the input and output formats, and execute the model.

4.5 Segmentation and recognition process

SSD predicts bounding boxes (segmentation) and class probabilities (recognition) in a single evaluation from the neural network, which greatly reduces per-frame processing time compared to older object detection frameworks such as Faster R-CNN [8]. Processing time is further reduced by DSCs. Characters detected with probability greater than or equal to 0.5 are considered part of the final character string.

Table 3 Test result statistics from the Caltech Cars [12] dataset

per-plate recognition accuracy	91.27%
per-character segmentation accuracy	96.46%
per-character recognition accuracy	96.23%
per-character mean precision	100.00%
per-character mean recall	96.46%
per-character mean F1-score	97.70%
mean average precision (at 0.5 confidence)	0.9933
average intersection-over-union for segmented chars, compared to labelled ground-truth bounding boxes	0.8878

Table 4 Test result statistics from the UCSD-Stills [13] dataset

per-plate recognition accuracy	97.59%
per-character segmentation accuracy	99.79%
per-character recognition accuracy	99.79%
per-character mean precision	100.00%
per-character mean recall	99.79%
per-character mean F1-score	99.88%
mean average precision (at 0.5 confidence)	1.00
average intersection-over-union for segmented chars, compared to labelled ground-truth bounding boxes	0.8730

Table 5 Speed result of our proposed system

average per-plate processing time (CPU)	59 ms
average per-plate processing time (CPU + OpenVINO)	14 ms
average per-plate processing time (Raspberry Pi 3 + OpenVINO + NCS2)	66 ms

5 Results and comparisons

5.1 Evaluation metrics

We chose the following metrics to evaluate our work: accuracy per-license plate, per-character segmentation, and per-character recognition; precision per-character segmentation; recall per-character segmentation; F1-score per-character segmentation; and processing time using a variety of hardware. Our focus was on achieving high per-plate recognition accuracy and low processing time.

Precision measures how well detected regions agree with corresponding ground-truth. It was calculated per-image by taking the number of predicted regions that had a corresponding ground-truth and dividing it by the total number of predicted regions.

Recall measures how well ground-truth regions were detected. It was calculated by taking the number of predicted regions that had a corresponding ground-truth and dividing it by the total number of ground-truths.

F1-score is the harmonic mean of precision and recall.

Accuracy was calculated as the number of correct detections divided by the total number of predictions.

Classification accuracy was calculated as the number of correctly classified regions divided by the total number of character segmentations.

A plate was considered recognised if all its constituting characters were detected and classified correctly. Plate recognition accuracy was therefore the number of recognised plates divided by the total number of seen plates (one hundred and twenty six for the Caltech Cars [12] and two hundred and ninety-one for the UCSD-Stills dataset).

Mean average precision (mAP) was calculated using the PASCAL VOC 2012 method. mAP takes the sum of the precisions over all character classes and divides it by the number of classes (36).

Processing time was calculated as the amount of time from direct image input to prediction output. Further descriptions can be found in Section 5.2.

5.2 Results

We tested our model on the Caltech Cars [12] and UCSD-Stills [13] datasets using license plate images cropped from the vehicle images in the datasets; variability in the area of the (unscaled) license plate region inputs was mitigated by resizing to 300×300 px. Our primary tests were conducted on an Ubuntu 18.04 LTS computer using an Intel Xeon CPU with 12 cores (2.60 GHz per core). These experiments were implemented in Python 3.6.5 using the libraries Tensorflow v. 1.8 and OpenCV v. 3.4.0. Accuracy tests were taken at a bounding box confidence of at least 0.5. Timing measurements were taken using *time.process_time()* from Python's *time* library.

Additional tests for processing time were conducted in the following platforms/environments:

- The same Ubuntu 18.04 computer and CPU but with OpenVINO model optimisation in a C++ implementation.
- A Raspberry Pi 3 and OpenVINO implementation (further described in Section 5.3).

Our results are contained in Tables 3–5.

About 86.5% (the 'majority') of plates in the Caltech Cars dataset had white backgrounds and dark characters. Within this majority, our DL solution segmented and recognised most characters correctly. When applied to the few other styles of plates, however, results were less consistent, with anywhere from one to more than half to all the available characters recognised. This represents likely class imbalance in terms of the colours of the

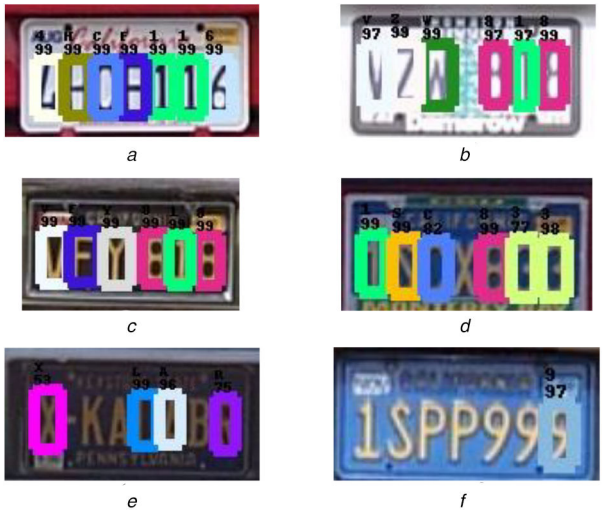

Fig. 6 *Examples of characters segmented and recognised via our DL solution on the Caltech Cars [12] dataset*
(a)–(c) Successfully recognised, ***(d)–(f)*** Partially recognised
Recognised characters and network confidence scores are above each detected bounding box

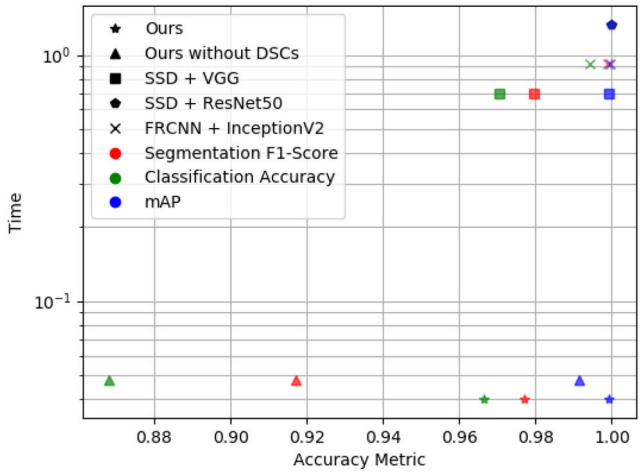

Fig. 7 *Model performance versus processing time for varying performance metrics and models on the Caltech Cars dataset [12]*

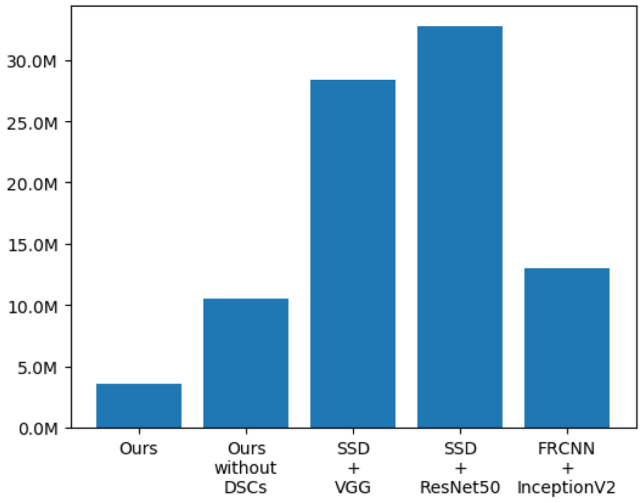

Fig. 8 *Number of parameters of various models*

plate-constituting characters. See Fig. 6 for examples of recognised characters on plates.

Regardless, our final solution achieved a per-plate accuracy of 91.27%, a per-character segmentation accuracy of 96.46%, and a per-character recognition accuracy of 96.23% on [12].

License plate types were rather uniform across the UCSD-Stills dataset, with 99.3% of plates having dark characters on a lighter background. However, the worst recognition performance, in which only three characters were recognised, occurred on an incredibly blurred and partially occluded 'majority' plate. Our model achieved a per-plate accuracy of 97.59%, a per-character segmentation accuracy of 99.79%, and a per-character recognition accuracy of 99.79% on [13].

In general, we found that a longer training time contributed to a higher recognition accuracy but degraded the accuracy of segmented region detection, meaning some characters could be missed.

Using OpenVINO with the Xeon CPU contributes to a 76% processing time reduction compared to using simply the CPU.

5.3 Comparisons to popular deep learning models

We sought to replace the VGG feature extractor in the SSD object detector with a faster model with minimal accuracy loss. Compared to fine-tuning on the original SSD + VGG implementation, our solution had no decrease in mAP for predictions >0.5 confidence, with a reduction in processing time of 94.26% using the CPU.

For a brief discussion on accuracy with increasing network depth and (CPU) processing time, we compare the mAP, segmentation F1-score, and recognition accuracy of our network against the following other object detection + feature extractor networks trained and tested using a process and methodology similar to ours: ours trained without using DSCs; SSD + VGG; SSD + ResNet50; and FR-CNN + InceptionV2. Timing measurements were taken using *time.perf_counter()* from Python's *time* library. Performance comparisons can be found in Fig. 7. Network parameter counts are shown in Fig. 8.

As seen in Figs. 7 and 8, our model performs nearly as well as the models with much higher parameters and processing times with minimal reductions in accuracy. For example, SSD + ResNet50 achieves 100% across all performance categories. It surpasses our model by only 2.3% segmentation F1-score, 3.77% classification accuracy, and 0.0067 mAP. However, our processing time is more than 33 × faster, and much more suitable to real-time applications.

Interestingly, our model performs better than the version without DSCs. This could be due to overfitting with the increased parameter count and otherwise similar network depth.

5.4 Comparisons to other research work

Comparisons to other research works that tested using the Caltech Cars [12] and UCSD-Stills [13] datasets may be found in Table 6.

On [12], our model is 1.5–2% more accurate than [23] or [25], and 0.13% more accurate than [29]. On [13], our model is about 2.5% more accurate than [29].

Although [25] has a lower parameter count, its segmentation is performed without DL via pure computer vision. Neither [23] nor [25] report processing time.

The processing time of [29] is about 14 ms for a plate with seven characters, as would be for the vast majority of plates in [12, 13]. This is the same as our CPU system optimised using OpenVINO. However, for inference, [29] used a high-end GPU – impractical for a low-cost edge system.

An interesting difference between our model and [23] or [25] is the amount of training data. Polishetty *et al.* [23] reports using almost 32× the number of plates, and [25] more than 18× the number of characters, used to train our model. Although this difference is less prominent comparing against [29], our system still uses less than half of the amount of data used to train [29]. This shows our model and training method is effective despite less data.

5.5 Implementation on a Raspberry Pi 3 with Neural Compute Stick 2 and OpenVINO

Many ALPR systems generate predictions from locally-captured video using an external server. Especially when the server is located far from the setup, high bandwidth is required between each camera and the server, which increases the cost of the system

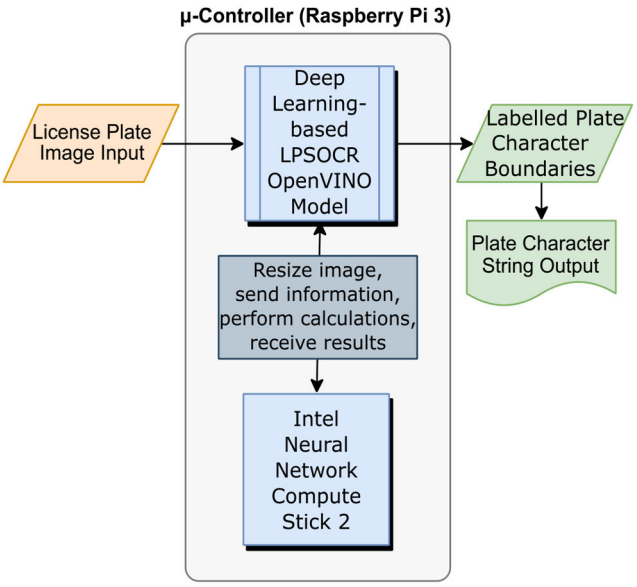

Fig. 9 *Simple block diagram of Raspberry Pi 3 + OpenVINO + Intel Neural Compute Stick 2 embedded system*

Table 6 Comparisons to other academic works, by dataset

Work	Per-character OCR accuracy, %		Processing time per-image, ms	Size of dataset used in training process	Parameters
	Caltech cars [12]	UCSD-stills [13]			
ours	96.23	99.79	14 ms (CPU + OpenVINO)	2845 plates, or >19,000 characters	~3.4M
modified YOLO [29]	96.1 (without heuristics)	97.3 (without heuristics)	~14 ms (high-end GPU)	6205 plates	~3.1M
7-layer CNN [23]	94.28	N/A	none given	>90,000 plates	unclear from paper
curve joining + 6-layer CNN [25]	94.8	N/A	none given	~368,000 characters	~690,000

to run in real time. To improve ALPR, we propose a low-cost real-time solution operating at the edge.

Our proposed system is comprised of a Raspberry Pi 3 and an Intel NCS2. The Raspberry Pi 3 has a quad-core 1.2 GHz CPU and 1 GB of RAM. Our object detection network is implemented on the NCS2's low-power chip, capable of 1 trillion operations per second. The inference software is written in C++. A simple block diagram for the proposed system can be found in Fig. 9.

The inference time of our proposed system, averaged over 1000 iterations, is 66 ms (15 FPS) for a license plate resized to 300 × 300 px. This makes our system well-suited for low-cost real-time edge ALPR applications. Using OpenVINO and the NCS2 represents an only 7 ms increase in inference time compared to using a 12-core (2.60 GHz per core) CPU. For a plate with seven characters, as typical in California, this represents ~only a 1 ms increase per character.

6 Conclusions and future work

As automated ITS applications become more ubiquitous in society, research into creating fast, accurate, and efficient ALPR solutions becomes necessary. DL, with its demonstrated potential for state-of-the-art computer vision performance, is an excellent candidate for creating ALPR solutions.

We used a LPSOCR DL system comprised of a feature extractor with DSCs and linear bottlenecks, to preserve accuracy and lessen parameter count, in conjunction with SSD architecture [11] that itself uses DSCs. Per-character accuracy on the Caltech Cars [12] dataset is 96.46% for segmentation and 96.23% for recognition. Per-character accuracy on the UCSD-Stills [13] dataset is 99.79% for both segmentation and recognition (Table 6).

With our proposed low-cost Raspberry Pi 3 and Intel NCS2 with OpenVINO embedded system, our model has a processing time of 66 ms. Using the Xeon CPU, it has a processing time of 59 ms. With the CPU and OpenVINO, it has a processing time of only 7 more ms – or 1 ms more per character, on average – than the CPU alone.

We foresee our solution being integrated into a low-cost edge end-to-end ALPR system.

Although we have proved the accuracy and potential for our embedded system, we believe our model inference would be more applicable to real-world situations with increased data variance. Thus, future work involves re-training with additional and varied (e.g. non-Californian, non-American) license plate character datasets. Additionally, we may try to simplify our architecture to lower its parameters.

7 Acknowledgments

This work is supported by Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grants and the NSERC Tri-Agency Canada Graduate Scholarships-Masters.

8 References

[1] Wang, Y., Lin, W., Horng, S.: 'A sliding window technique for efficient license plate localization based on discrete wavelet transform', *Expert Syst. Appl.*, 2011, **38**, (4), pp. 3142–3146

[2] Yépez, J., Ko, S.: 'Improved license plate localization algorithm based on morphological operations', *IET Intell. Transp. Syst.*, 2018, **12**, (6), pp. 542–549

[3] 'OpenVINO Toolkit'. Available at https://software.intel.com/en-us/openvino-toolkit, accessed June 2019

[4] Bengio, Y.: '*Learning deep architectures for AI, now foundations and trends*' (Now Publishers, Montreal, Canada, 2009, 1st edn.)

[5] Angara, N.: 'Automatic License Plate Recognition Using Deep Learning Techniques'. MSc Thesis, University of Texas at Tyler, 2015

[6] Kurpiel, D.: 'Localização de placas veiculares em vídeo usando redes neurais convolucionais profundas'. BSc thesis, Universidade Tecnológica Federal Do Paraná, 2018

[7] Simonyan, K., Zisserman, A.: 'Very deep convolutional networks for large-scale image recognition', arXiv:1409.1556v6, accessed August 2018

[8] Ren, S., He, K., Girshick, R., *et al.*: 'Faster R-CNN: towards real-time object detection with region proposal networks', *IEEE Trans. Pattern Anal. Mach. Intell.*, 2017, **39**, (6), pp. 1137–1149

[9] Dai, J., Li, Y., He, K., *et al.*: 'R-FCN: object detection via region-based fully convolutional networks'. Advances in Neural Information Processing Systems, Barcelona, Spain, 2016, pp. 379–387
[10] Yépez, J., Castro-Zunti, R., Ko, S.: 'Deep learning-based embedded license plate localisation system', *IET Intell. Transp. Syst.*, 2019, **13**, (10), pp. 1569–1578
[11] Liu, W., Anguelov, D., Erhan, D., *et al.*: 'SSD: single shot MultiBox detector', Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Amsterdam, Netherlands, 2016, vol. 9905, pp. 21–37
[12] Caltech: 'Computational vision: archive', Available at http://www.vision.caltech.edu/html-files/archive.html, accessed August 2018
[13] Dlagnekov, L., Belongie, S.: 'UCSD/calit2 car license plate, make and model database', available at http://vision.ucsd.edu/belongie-grp/research/carRec/car_data.html, accessed October 2019
[14] Hongyao, D., Xiuli, S.: 'License plate characters segmentation using projection and template matching'. Int. Conf. on Information Technology and Computer Science, Kiev, Ukraine, 2009, pp. 524–537
[15] Kasaei, S., Kasaei, S.: 'Extraction and recognition of the vehicle license plate for passing under outside environment'. Proc. European Intelligence and Security Informatics Conf., Athens, Greece, 2011, pp. 234–237
[16] Abderaouf, Z., Nadjia, B., Saliha, O.: 'License plate character segmentation based on horizontal projection and connected component analysis'. World Symp. on Computer Applications & Research, Sousse, Tunisia, 2014, pp. 1–5
[17] Karthikeyan, V., Vijayalakshmi, V., Jeyakumar, P.: 'License plate detection using morphological operators', Available at http://www.academia.edu/2554097/License_Plate_Segmentation_Based_on_Connected_Component_Analysis, accessed October 2018
[18] Yoon, Y., Ban, K., Yoon, H., *et al.*: 'Blob extraction based character segmentation method for automatic license plate recognition system'. IEEE Int. Conf. on Systems, Man, and Cybernetics, Anchorage, United States of America, 2011, pp. 2192–2196
[19] Jeffrey, Z., Ramalingam, S., Bekooy, N.: 'Real-time DSP-based license plate character segmentation algorithm using 2D haar wavelet transform', in Baleanu, D. (Ed.): '*Advances in wavelet theory and their applications in engineering, physics and technology*' (IntechOpen, Welwyn Garden City, UK, 2012, 1st edn.), pp. 3–22
[20] Bulan, O., Kozitsky, V., Ramesh, P., *et al.*: 'Segmentation- and annotation-free license plate recognition with deep localization and failure identification', *IEEE Trans. Intell. Transp. Syst.*, 2017, **18**, (9), pp. 2351–2363
[21] Li, J., Wang, P., You, M., *et al.*: 'Reading car license plates using deep neural networks', *Image Vis. Comput.*, 2018, **72**, pp. 14–23
[22] Hendry Chen, R.-C.: 'Automatic license plate recognition via sliding-window darknet-YOLO deep learning', *Image Vis. Comput.*, 2019, **87**, pp. 47–56
[23] Polishetty, R., Roopaei, M., Rad, P.: 'A next-generation secure cloud-based deep learning license plate recognition for smart cities'. IEEE Int. Conf. on Machine Learning and Applications, Anaheim, United States of America, 2016, pp. 286–294
[24] National Technical University of Athens, Greece: 'Medialab LPR database', http://www.medialab.ntua.gr/research/LPRdatabase.html, accessed August 2018
[25] Selmi, Z., Halima, M., Alimi, A.: 'Deep learning system for automatic license plate detection and recognition'. Int. Conf. on Document Analysis and Recognition, Kyoto, Japan, 2017, pp. 1132–1138
[26] Montazzolli, S., Jung, C.: 'Real-time Brazilian license plate detection and recognition using deep convolutional neural networks'. Conf. on Graphics, Patterns and Images, Niteroi, Brazil, 2017, pp. 55–62
[27] Mottaghi, R., Chen, X., Liu, X., *et al.*: 'The role of context for object detection and semantic segmentation in the wild'. IEEE Conf. on Computer Vision and Pattern Recognition, Columbus, United States of America, 2014, pp. 891–898
[28] Laroca, R., Severo, E., Zanlorensi, L., *et al.*: 'A robust real-time automatic license plate recognition based on the YOLO detector'. Int. Joint Conf. on Neural Networks (IJCNN), Rio de Janeiro, Brazil, 2018
[29] Laroca, R., Zanlorensi, L., Gonçalves, G., *et al.*: 'An efficient and layout-independent automatic license plate recognition system based on the YOLO detector', arXiv.org: 1909.01754v2, accessed October 2018
[30] Abedin, M., Nath, A., Dhar, P., *et al.*: 'License plate recognition system based on contour properties and deep learning model'. IEEE Region 10 Humanitarian Technology Conf., Dhaka, Bangladesh, 2017, pp. 590–593
[31] How, D., Sahari, K.: 'Character recognition of Malaysian vehicle license plate with deep convolutional neural networks'. IEEE Int. Symp. on Robotics and Intelligent Sensors, Tokyo, Japan, 2016, pp. 1–5
[32] Campos, T., Babu, B., Varma, M.: 'Character recognition in natural images'. Proc. of the Int. Conf. on Computer Vision Theory and Applications, Lisbon, Portugal, 2009, pp. 273–280
[33] Sandler, M., Howard, A., Zhu, M., *et al.*: 'Mobilenetv2: inverted residuals and linear bottlenecks'. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, United States of America, 2018, pp. 4510–4520
[34] He, K., Zhang, X., Ren, S., *et al.*: 'Deep residual learning for image recognition'. Proc. of the IEEE Computer Society Conf. on Computer Vision and Pattern Recognition, Las Vegas, United States of America, 2016, pp. 770–778
[35] Yu, D., Seltzer, M.: 'Improved bottleneck features using pretrained deep neural networks'. Interspeech, Florence, Italy, 2011, pp. 237–240
[36] tzutalin: 'Labelimg', available at https://github.com/tzutalin/labelImg, accessed July 2018