Received: 28 January 2021 | Revised: 8 April 2021 | Accepted: 4 May 2021

DOI: 10.1002/int.22471

RESEARCH ARTICLE

WILEY

License plate recognition using neural architecture search for edge devices

Jithmi Shashirangana[1] | **Heshan Padmasiri**[1] | **Dulani Meedeniya**[1] | **Charith Perera**[2] | **Soumya R. Nayak**[3] | **Janmenjoy Nayak**[4] | **Shanmuganthan Vimal**[5] | **Seifidine Kadry**[6]

[1]Department of Computer Science and Engineering, University of Moratuwa, Moratuwa, Sri Lanka

[2]School of Computer Science and Informatics, Cardiff University, Cardiff, UK

[3]Amity School of Engineering and Technology, Amity University Uttar Pradesh, Noida, India

[4]Department of CSE, Aditya Institute of Technology and Management (AITAM), Tekkali, India

[5]Department of CSE, Ramco Institute of Technology, Rajapalayam, India

[6]Department of Applied Data Science, Noroff University College, Kristiansand, Norway

Correspondence
Dulani Meedeniya, Department of Computer Science and Engineering, University of Moratuwa, Moratuwa 10400, Sri Lanka.
Email: dulanim@cse.mrt.ac.lk

Present address
Department of CSE, University of Moratuwa, Moratuwa 10400, Sri Lanka.

Abstract

The mutually beneficial blend of artificial intelligence with internet of things has been enabling many industries to develop smart information processing solutions. The implementation of technology enhanced industrial intelligence systems is challenging with the environmental conditions, resource constraints and safety concerns. With the era of smart homes and cities, domains like automated license plate recognition (ALPR) are exploring automate tasks such as traffic management and fraud detection. This paper proposes an optimized decision support solution for ALPR that works purely on edge devices at night-time. Although ALPR is a frequently addressed research problem in the domain of intelligent systems, still they are generally computationally intensive and unable to run on edge devices with limited resources. Therefore, as a novel approach, we consider the complex aspects related to deploying lightweight yet efficient and fast ALPR models on embedded devices. The usability of the proposed models is assessed in real-world with a proof-of-concept hardware design and achieved

Abbreviation: ABC, a black cat; DEF, does not ever fret; GHI, goes home immediately.

Jithmi Shashirangana and Heshan Padmasiri contributed equally to this work.

competitive results to the state-of-the-art ALPR solutions that run on server-grade hardware with intensive resources.

KEYWORDS
automatic license plate recognition, edge devices, intelligent systems, neural architecture search

1 | INTRODUCTION

In modern society, intelligent systems (IS) have been expanded phenomenally for the task of information processing. The fusion of the internet of things (IoT) and artificial intelligence (AI) technology has enabled us to develop intelligent information processing solutions at an industrial level. However, some of the key challenges of designing and implementing such industrial IS systems are data collection, storage management, and security concerns in resource-constrained safety-critical applications, and so on. Therefore, it is important to address the associated constraints and frontiers in deploying intelligent systems with IoT.

The problem of automatic license plate recognition (ALPR) has gained a lot of attention mainly because of its various practical applications like traffic law enforcement, collecting toll payments, and managing exit and entrance in vehicle parks, and so on. Over the years, many studies have tackled the problem of ALPR with various techniques like using classical computer vision techniques and lately with modern deep learning models.[1] But still, most of these prevailing solutions only work under controlled environments and conditions. Especially modern deep learning solutions for ALPR are generally computationally intensive and unable to run on edge devices with limited computational resources. Therefore, there is a need to build lightweight and fast models that are implementable on embedded devices.

There are certain reasons for implementing ALPR solutions to run on edge devices. First, ALPR systems are widely employed in scenarios where there is a need to prevent some crime or fraud activities and, therefore, it is highly expected to have maximum security in them. For instance, an ALPR system embedded in a surveillance system may need to send security footage over the internet, and practically this raises some privacy concerns. Also, since the system is exposed to the internet, it is open to other malicious attacks such as hacking which will diminish the reliability of the system. These existing challenges can be mitigated if the inference is performed on-site within a cost and power-efficient edge device. Although there exist studies on license plate recognition on edge devices,[2-4] the obtained accuracy are low in comparison to the state-of-the-art solutions. To the best of our knowledge, there has been no previous effort for license plate recognition purely on edge devices with accuracy competitive with the solutions designed to run on server-grade hardware.

The proposed research presents an optimized decision support solution for ALPR that works purely on edge devices with limited resources at nighttime. We have considered some of the commonly associated complexity aspects related to ALPR like plate variations across regions, environmental impacts such as changing illuminations, nighttime operation without additional illumination, and other challenging weather conditions as well. However, one of the main focuses of our research is to reduce the computational complexity of ALPR models and their expensive memory usage. Although we designed our models less complex for embedded

devices, still the system evaluation has shown a recognition accuracy competitive to the state-of-the-art ALPR solutions designed to run on server-grade hardware. Moreover, we also have evaluated the usability of the proposed models in real-world with a proof-of-concept hardware design for license plate recognition. In the future, this can also be used as a smart city solution for law enforcement agencies to enhance their capabilities and enforcement in controlling traffic and detecting fraud activities.

Although, neural networks for license plate recognition is a well-explored area for the daytime images with the server-grade hardware specification, we have provided the following main contributions in this study:

- A novel approach for a hardware-efficient ALPR solution for edge devices.
- Designed and developed neural architecture search (NAS) to optimize models for hardware platforms with limited resources.
- The synthetic data generation process for nighttime license plate recognition to mitigate the issue with the scarcity of a large and diverse nighttime license plate data set for training the learning models.
- Guideline to adopt the developed models for ALPR process towards real-world deployments.

2 | BACKGROUND

2.1 | Evolution of license plate recognition approaches

2.1.1 | License plate recognition techniques

Over time, an extensive literature has developed on ALPR. These methods can be broadly classified into two categories based on the number of main stages in the plate recognition process.[1] Multistage license plate recognition is the widely used approach and it separates the license plate recognition process into two main stages known as license plate detection and recognition. While there are also few successful attempts in developing single-stage license plate recognition systems and currently, they serve as the state-of-the-art models in the ALPR domain. There is a wide choice of techniques for solving the ALPR problem, where many early studies have used classical image processing techniques. Recently, computer vision and deep learning-based techniques have been using to detect and recognize license plates.[1] These techniques make use of global features of a license plate such as the shape, color, texture, and presence of characters and can be summarized as follows.

- Edge-based detection techniques use the rectangular shape information of the license plate. The Sobel filter is one of the most used techniques in image processing to detect license plates. In Reference [5], they have used a Sobel filter to extract only the vertical edges of the plate as the horizontal edges could lead to confusion between the car bumper and the plate edges. Rasheed et al.[6] have used a canny edge detector to extract all the edges in the image and then identifies the plate edges using a Hough transformation. Edge-based methods are much faster to compute but sensitive to unwanted edges in the images and have difficulty in detecting license plates that are inclined or deformed.
- Color-based detection techniques use the fact that the license plate has a different color to the vehicle color. Ashatri et al.[7] have proposed color geometric templates to localize license plates.

Chang et al.[8] have examined a color edge detection method to differentiate the foreground and background regions in the image. They used this to detect Taiwanese license plates. While color based approaches are robust against inclined or deformed license plates, they are also sensitive to illumination variations and therefore they are seldomly used alone.

- Texture-based approaches use unique pixel intensity distribution in the license plate region to detect the license plate. Yu et al.[9] have proposed a method that uses a combination of Wavelet transform and empirical mode component analysis to locate the license plate. Giannoukos et al.[10] have introduced a novel technique called sliding concentric window to detect the license plate.
- Character-based approaches use the fact that the license plate consists of characters and locate the region with characters as a possible plate region. Zhou et al.[11] have modeled the license plate detection problem as a visual matching problem and created principal visual words for each legal character in the license plate to detect the license plate.
- Statistical classifiers like support vector machines (SVM) and cascaded classifiers are also used is several studies on ALPR. However, with the rapid advancement of the deep learning paradigm, most of the statistical models were replaced with more powerful deep learning models.
- Considering deep learning models in the ALPR context, several studies[12-14] have used state-of-the-art single-shot object detectors like You only look once (YOLOv3)[15] to detect the license plate by generalizing the license plate detection to an object detection task. However, Residual Pyramid Network (RPNet)[16] and Tuple-based End-to-end (TE2E) loss function[17] were also rooted in deep learning techniques and currently considered as the state-of-the-art models in the ALPR context.

2.1.2 | Related work on license plate recognition

Most of the existing ALPR systems[5-10] that are built with traditional computer vision techniques follow the multistage approach. Conversely, the single-stage license plate recognition models, RPNet and TE2E, perform license plate recognition as a single task without having to localize the plate beforehand. These models use a single deep neural network that is trained end-to-end for recognizing license plates in a single forward computation. Both of these methods use stacked convolutional layers which act as feature extractors and are then used to generate region of interest (RoI) proposals. Then these RoIs are pooled and used to recognize the license plate.

Li et al.[17] were the first to create such a single-stage license plate recognition model. In their work, they have adopted the existing VGG16 (Visual Geometry Group from Oxford) convolutional neural network model for low-level feature extraction.[18] The VGG16 consists of 13 convolutional layers, 5 max-pooling layers, and fully connected layers. In the proposed TE2E model, they have eliminated the fully connected layers and given the fact that the license plate carries only a smaller portion in the entire image. This method has used two max-pooling layers instead of five to make sure that no important information vanished. Avoiding the need to perform character segmentation, they have used bidirectional RNNs for license plate recognition. This allows its network to handle arbitrary length license plates.

The RPNet model[16] is composed of two subnetworks for plate detection and LP number prediction. The plate detection subnetwork is stacked with 10 convolutional layers, and three fully connected layers to predict the bounding box of the license plate directly. The output

feature maps from multiple layers 1, 3, and 5 are then sent to multiple classifiers though RoI pooling layers that extract feature maps of interest. Xu et al.[16] have found that using separate subnetworks made up of convolution neural networks for each character in the license plate to be more accurate. Therefore, given that the license plate has a fixed number of characters, the recognition subnetwork has simpler classifiers for each character in the license plate. However, all the training and evaluation tasks for RPNet are accomplished on extremely powerful server-grade hardware and the model is trained with a large and diverse data set. Also, the RPnet is claimed to be 20 times faster than the TE2E model also shallower due to the use of simpler feature extractors in the detection subnetwork. Therefore, this model currently gives considerably high accuracy over many other ALPR models.

Most of the related work in the ALPR context share a common limitation, as they require high computational resources to achieve maximum accuracy in the recognition process. Due to this trade-off between the accuracy and the memory, most of the solutions have failed to achieve both and instead, are focused on improving one of the aspects. However, overcoming this challenge, few successful attempts have been made to develop lightweight ALPR models that can execute on edge devices with limited resources. A study by Alborzi et al.[19] have proposed a lightweight ALPR model that is implementable purely on embedded devices like Raspberry Pi3. They have used a combination of MobileNet feature extractor with fewer parameters along with a Single-shot detection model to achieve the least usage of memory for the detection stage. Also, for the character recognition, they utilized a powerful but yet, computationally affordable network which is LPRNet[20] and the authors report an end-to-end accuracy of 79.86%. Another study by Izidio et al.[21] have proposed a deep learning based system for embedded systems with a miniature version of the state-of-the-art YOLOv3 algorithm for plate detection and another convolutional neural network (CNN) model for character recognition. The proposed system was implemented to run on a Raspberry Pi3 with an overall recognition rate of 98.43%.

2.2 | Neural architecture search

2.2.1 | NAS process

Today, deep neural networks, especially CNNs, have become the de facto approach for solving computer vision tasks. This is mainly due to their significant accuracy and robustness compared to other approaches. However, designing a neural network for a given task is not necessarily a straightforward task. Since designing high-performance state-of-the-art neural architectures manually requires expert knowledge and a time-consuming process, recently NAS methods have attracted a lot of attention in automating the process of architecture modeling. These NAS methods lead to discover better, faster and cost-efficient solutions for tasks like image classification and object detection. Therefore, rather than selecting an architecture based on the previous experiments on a similar domain, NAS methods are used to explore some novel, potential architectures for a specific task.[22] Figure 1 summarizes the general framework of the NAS process. Generally, a NAS process is comprised of a set of predefined operation sets called search space and a search strategy that helps to obtain a set of candidate network architectures by exploring the search space.[22] Then in the performance estimation phase, the candidate architecture is trained using the training data set and ranked based on their performance on the validation data. According to the ranking information, the search strategy

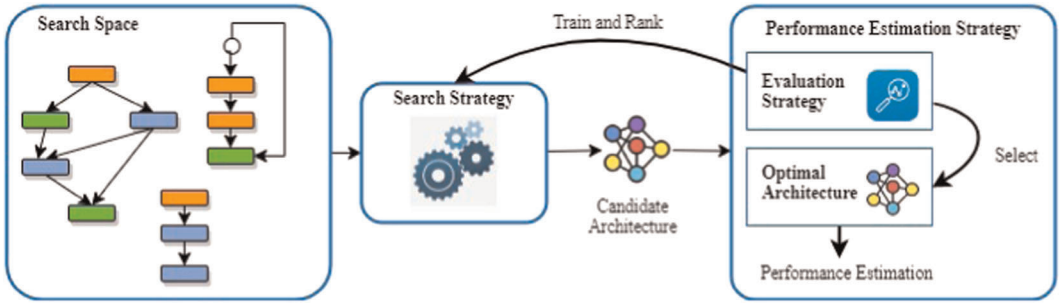

FIGURE 1 The general framework of the neural architecture search process [Color figure can be viewed at wileyonlinelibrary.com]

is adjusted and finally the most optimal architecture is selected. Then the final optimal architecture is evaluated on the test set for performance.

2.2.2 | NAS strategies

Several architecture search strategies have been proposed and they can be considered as either black-box optimization strategies such as reinforcement learning, evolutionary programming, and Bayesian optimization or differential architecture search strategies like differentiable architecture search (DARTS)[23] and partially connected differentiable architecture search (PC-DARTS).[24]

In this study, we define architecture search as given in [25],

$$\min_{a \in A} \min_{w_a} L(a, w_a) \tag{1}$$

where A is the candidate architecture space, w_a is the weights after training a, and L is the loss function.

Since A is a discrete space in black-box optimization strategies, we cannot use techniques such as gradient descent to directly optimize it. However, it is possible to optimize over discrete space using methods such as reinforcement learning, evolutionary programming and Bayesian optimization. These methods are called black-box optimization strategies because they consider architecture a as a black-box. One of the earliest attempts that managed to reach performance competitive with human designed architectures was by Zoph and Le[26] using reinforcement learning.

While black-box optimization strategies have achieved and sometimes even surpassed human designed architectures, the biggest hurdle for their wide spread use is the massive computational cost. While various optimization for these strategies such as limiting the search space to follow structures in order of increasing complexity,[27] weight sharing,[28] and performance prediction[29] instead of training the architecture have been proposed, they remain prohibitively expensive to apply on most practical use cases. The main reason for this is that the black-box optimization strategies require a large number of architecture evaluations to find a and each evaluation require either partial or complete training of a neural network.

When compared to black-box optimizations, the differential architecture search methods have a major advantage of achieving competitive results with only fewer iterations by

optimizing using the gradient descent technique. However, gradient descent can only be used with differential architecture search strategies as they have a continuous search space.

While there had been previous attempts[30] at searching architectures withing continuous domain, DARTS[23] by Liu et al. is the first successful attempt we are aware of to fully search for an architecture. They represented the search space as a directed acyclic graph with input node x_0 and output node x_n. Each node is a tensor and each edge ($o_{(i,j)}$) connecting nodes x_i and x_j is an operator. Each node is connected to every one of its previous nodes and the value of node x_j is obtained by,

$$x_j = \sum_{i<j} o_{(i,j)}(x_i) \tag{2}$$

They relaxed the categorical choice of $o_{(i,j)}$ into a softmax over all possible operations. This relaxes A into a continuous search space where both weights w_a and a can be optimized in a single network using bilevel optimization strategy.[31] Thus the final architecture can be obtained by simply using argmax on each edge. This search strategy allowed them to achieve a state-of-the-art performance significantly quicker than black-box optimization strategies.

2.2.3 | NAS-related studies

The license plate detection and the plate recognition tasks of an APLR system can be generalized into object detection and character recognition problems. In the object detection context, there is an increasing demand for highly accurate models but with limited cost of computational resources. The preliminary work on applying the NAS algorithm for object detection was NAS-FPN[32] and Auto-FPN.[33] Also, Google's research on EfficientDet[34] proposed a new family of object detectors based on their previous work on EfficientNet[35] which was discovered by exploring a NAS algorithm. While adapting to resource constraints in limited-memory devices, EfficientDet models show state-of-the-art accuracy in object detection and also claim to be 9 times smaller than the prior state-of-the-art object detectors.

Another study by Du et al.,[36] have proposed a novel model called SpineNet which also used a NAS algorithm to discover an effective scale-permuted object detection model. Chen et al.[37] also designed a better backbone framework for object detection using evolutionary search algorithms for architecture search. In addition to NAS on object detection, some recent works attempt to develop NAS for other tasks, especially character recognition. In Reference [38], they proposed a memory-efficient CNN-based model for scene text recognition and applied a proxylessNAS strategy to enable end-to-end training. Another study by Zhao et al.[39] presented a method called AutoOCR based on an evolutionary computation NAS method for optical character recognition.

Moreover, the optimized architecture depends on feature external to the architecture such as the input size and hardware platform on which it runs.[25] However, this is difficult to take into account due to the size of the design space. Instead, designers create generic solution agnostic to input size and hardware which as a result is suboptimal per each input size and hardware design. While some attempts have been made to create hardware agnostic efficiency matrices to measure the efficiency irrespective of target hardware such as floating point operations count (FLOPs), since they do not take into account subtleties of each hardware design into account they can lead to an architecture that is worse than expected. An example would be the performance difference between NASNet[40] and MobileNetV1.[41] While Both have similar FLOP count latency of NASNet is worse than that of MobileNetV1 due to its fragmented cell structure.[42]

Further, the work done in Reference [25] have proposed a family of hardware-aware models called FBNets that are optimized for mobile platforms using a Differential Neural Architecture Search. Similarly, a recent study by Tan et al.[43] presented another set of convolutional neural networks for mobile devices with a reinforcement-learning based search algorithm. Another recent study by Zhang et al.[44] proposed a hardware-efficient model called HURRICANE using one-shot NAS methods and currently it is considered to have state-of-the-art performance in mobile platforms.

3 | METHODOLOGY

Most of the existing well-defined license plate recognition models such as RPNet are not designed to run on memory constraint embedded devices. Therefore, their operations are not optimized for different hardware accelerators in them. In this paper, we propose a family of neural networks for license plate recognition that execute purely on embedded devices using a novel approach.

Since one of the objectives of this study is to minimize the computational cost while maximizing the accuracy of the license plate recognition models, we have used a NAS to automate the architecture modeling for detection and recognition networks. In this study, two differential architecture search strategies namely PC-DARTS and FBNet are explored. The overall process with the NAS is shown in Figure 2.

Due to the computational costs involved with neural architecture search, first, we sampled a proxy data set from a synthetically generated nighttime data set which is detailed in Section 5.1.1 to convert RGB images to TIR images. For this, we randomly selected 10,000 images from the transformed nighttime images. Against this data set, we performed a NAS using two NAS strategies: PC-DARTS and FBNet. Then we trained and evaluated the resultant networks on the entire data set by manually optimizing the models as described in Section 3.4. At the end of this process, we obtained the neural networks for license plate detection and recognition.

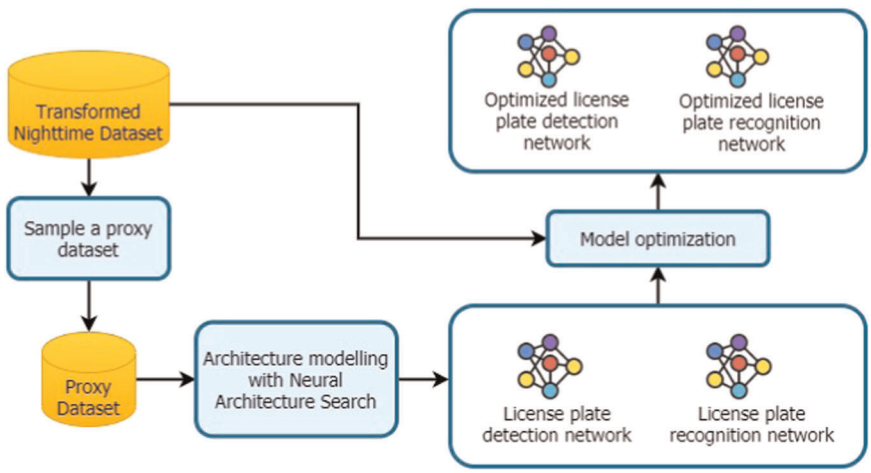

FIGURE 2 Neural architecture search process [Color figure can be viewed at wileyonlinelibrary.com]

3.1 | The search space

Generally, NAS strategies generate neural networks using a predefined set of operations called the search space.[22] These operations can be very granular such as a single convolutional layer with a given kernel size and the number of filters or coarser such as a block made up of several layers. The advantage of using more granular operations is that it makes the architecture search more flexible allowing more complicated architectures to be generated. However, this will take more time to converge the search results, and initial search results may be less than satisfactory.

A coarser search space can use "neural blocks" based on the existing understanding of the domain, which gives functional results with few iterations, as the search strategy starts with a more complex task-oriented initial architecture. However, using neural blocks are not as flexible as the more granular search spaces, which can discover those neural blocks on their own and further improve them, in the long run. Thus, the proposed methodology uses the coarser approach to reduce the search time and our work was inspired by four types of neural blocks as the set of operations: (1) RPNet blocks, (2) MobileNet blocks, (3) Inception blocks, and (4) Identity connections.

3.1.1 | RPNet blocks

As shown in Figure 3A, the proposed model is based on RPNet,[16] which gives state-of-the-art performance both in terms of latency and accuracy in the license plate recognition domain. It was decided to use the same kernel sizes and filter sizes that were used in the original RPNet model since the model has achieved impressive results with those selections. Therefore we used two kernel sizes 5×5 and 3×3 and filter sizes 48, 64, 128, 160, and 192. We also added two modifications to these neural blocks in the form of factorized convolutions and depth-wise separable convolutions as they consume less memory and in many cases have less latency compared with traditional convolution operations.

3.1.2 | MobileNet blocks

As shown in Figure 3B, license plate detection can also be modeled as an object detection problem and therefore we selected neural blocks from MobileNetV2[45] as it is one of the most

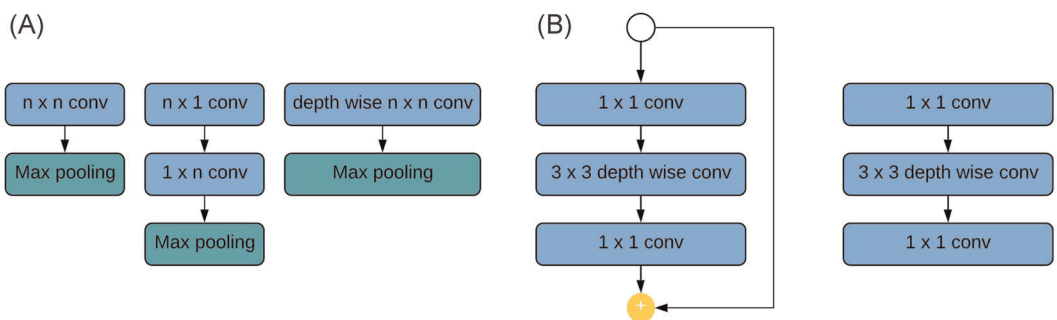

FIGURE 3 (A) RPNet blocks and (B) MobileNet blocks [Color figure can be viewed at wileyonlinelibrary.com]

commonly used choices as a backbone in many object-detection networks in resource constraint conditions. MobileNetV2 is a general-purpose computer vision neural network architecture that can run these deep networks on mobile devices or devices with low computational power with some adding benefits of less energy consumption and the reduced model size.

3.1.3 | Inception blocks

As shown in Figure 4, the models in the Inception family have been heavily useful in building custom classifiers that are optimized in both latency and accuracy. Instead of naively stacking larger convolutional operations to build deeper and deeper networks which a computationally expensive task to perform, the models in the Inception family introduce "wider" models rather than "deeper" models. Further, they added a new concept called factorization to reduce the dimension and in the meantime to reduce the problem of overfitting. Therefore, here we used the InceptionV4[46] B, D, and E blocks, especially considering some features like the balance of accuracy and less computational resource consumption.

3.1.4 | Identity connections

Identity connections as the name suggests performs identity transformation on the input tensor. In Facebook-Berkeley-Nets (FB-Net), this is used by the architecture search to create shallower networks. This is achieved by making the operation of a layer to be the identity operation so that layer can be removed without any effect on the model. In the case of PC-DARTS, this is done by connecting two tensors by identity connection which indicates both tensors to be the same. So the resulting network can be constructed with one less operation.

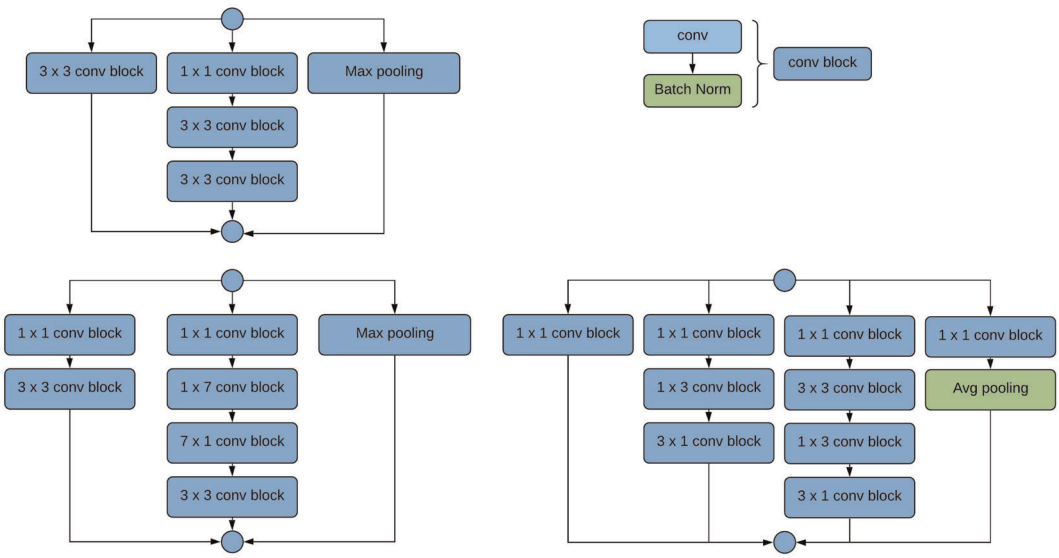

FIGURE 4 Inception blocks [Color figure can be viewed at wileyonlinelibrary.com]

3.2 | The search algorithms

This section gives a brief review of the two NAS strategies PC-DARTS and FBNet which were explored in this study to discover the neural network architectures for the plate detection and recognition models optimized for memory-constrained embedded devices.

3.2.1 | PC-DARTS implementation

PC-DARTS defines its stochastic super network as a directed graph, where vertices represent tensors and edges represent operation in the search space. We call the number of intermediate tensors as the depth of the network in our implementation. We used depth as a proxy for network complexity and memory consumption. Each tensor is connected to every one of its predecessors using all the operations in the search space.

The value of each tensor x_j can be defined using its predecessors as shown in Equation (3). Here we use the value of subscripts to represent the order of tensors and O represents the set of operations in the search space.

$$x_j = \sum_{i<j} \sum_{o \in O} \alpha_{(i,j,o)} o(x_i) \tag{3}$$

We call the value of $\alpha_{(i,j,o)}$ as the architecture weight of operation o for edge (i, j). These weights represent the probability of each connected tensor with its predecessor j using operation o. Therefore, we used a softmax distribution to represent these weights. We call the set of all such architecture weights as the architecture weights of the super network (w_α). Each operation such as convolution can have its own weights and the set of all such weights in the super network is known as the operation weights of the super network (w_θ). For brevity, we will refer to these weights as architecture weights and operation weights in the following discussion.

The final aim is to find an optimal architecture to obtain a given output (i.e., In this case either the bounding box of the license plate or the sequence representing characters in the license plate) from a given input (i.e., either the image directly from a camera or the image of the license plate). Since the architecture of the neural network is defined by w_α, we need a process to find the value of w_α that minimizes the loss L. We use the symbol w_{α^*} to represent this optimal w_α value. But since the loss value is depended not only on w_α but also on w_θ (because the output of each operation depends on their respective operation weights), we performed a bi-level optimization as defined in Equation (4).

$$\begin{aligned} &\min_{w_\alpha} \quad L\left(w_\alpha^*, w_\alpha\right) \\ &s.\ t \quad w_\alpha^* = argmin_{w_\theta} L(w_\theta, w_\alpha) \end{aligned} \tag{4}$$

Here we consider optimizing w_θ as the inner optimization task since the output of each operation is dependent on it and the suitability of each operation to connect given two tensors is depended on the output of that operation.

3.2.2 | FBNet implementation

Unlike PC-DARTS which followed a graph structure to represent its computations, FB-Nets stochastic super network is more similar to a typical feed-forward neural network. Each layer takes the output of the previous layer x_{i-1} and apply operation as shown in Equation (5) to obtain its output x_i. O is the set of all operations in the search space.

$$x_i = \sum_{o \in O} \alpha_{(i,o)} o(x_{i-1}) \tag{5}$$

We call the value $\alpha_{(i,o)}$, the architecture weight of layer i with respect to operation o. Set of all such weights is given by p_i as shown in Equation (6).

$$p_i = \alpha_{(i,1)} \; \forall \; o \in O \tag{6}$$

Since this p_i represents the probability of selecting a given operation as the operation of that layer we have represented this as a softmax distribution. We used a gumbel softmax for this as suggested by the original FBNet authors to reduce the training instability. Similar to PC-DARTS we define the set of all such p_i values as the architecture weights of the stochastic super network (w_α). We use the same definition as PC-DARTS to define the operation weights (w_θ) and also the same bi-level optimization algorithm presented in PC-DARTS to define the FB-Net algorithm.

To make this search process sensitive to characteristics of the target hardware platform, we used the same latency based modification to the loss function as in the original FBNet implementation. First, we create a latency table that contains the latency for executing each operation in the search space using the target hardware platform. Then we calculate the latency of the layer using p_i as shown in Equation (7).

$$LAT(p_i) = \sum_{o \in O} l\,at_o \alpha_{(i,o)} \tag{7}$$

Here $l\,at_o$ is the latency of operation o taken from the latency table. Then the latency of the super network ($LAT(w_\alpha)$) is obtained by summing the latency values of all the layers.

Then we define a new loss function L^* as shown in Equation (8). Here a and b are training hyperparameters. Since this loss is still differentiable with respect to both w_θ and w_α we can replace $L(w_\theta, w_\alpha)$ with this new $L^*(w_\theta, w_\alpha)$ in bi-level optimization algorithm to obtain a hardware sensitive architecture search process.

$$L^*(w_\theta, w_\alpha) = L(w_\theta, w_\alpha) \cdot alog(LAT(w_\alpha))^b \tag{8}$$

3.3 | NAS design process

The license plate detection process is modeled as a regression problem, where the model predicts a vector of four values, two representing the x and y coordinates of the center point of the bounding box and the other two representing the height and width of the bounding box. The license plate recognition process is explored in two variations.

1. A design similar to the TE2E model, where a single model predicts all the characters in the image. Here, a single network is trained from end-to-end that simulates the two-stage process of runtime and verification during training. The single model design shares parameters when recognizing each character, thus have less memory consumption.
2. The second approach is based on the design of the Roadside Parking Net (RPNet) model, where there exist a separate subnetwork for each character in the license plate. The memory consumption of the separate subnetworks is higher as the separate subnetworks cannot share the parameters.

We used root mean square error for license plate detection and cross-entropy for license plate recognition as the optimization loss functions. Also, a cosine annealing learning rate scheduler[47] was employed to control the variable learning rate that was used. Then we trained the stochastic super networks that are explained in the PC-DART and FBNet implementations in Section 3.2.1 and Section 3.2.2, respectively, and the obtained optimal architectures.

3.4 | Model optimization on the original data set

In general, one of the limitations of NAS is the computational cost associated with the process.[22] The process consumes more time to generate the best model when the data set is large in scale. To reduce the time and computational requirements of the architecture search process, while maintaining efficiency, it is suggested to use a reliable data proxy for NAS. This proxy data set, D_{proxy} is a subset of the original data set, $D_{original}$ and can be 10–20 times smaller compared to the original data set. In this study, we performed the architecture search on a proxy data set using a method suggested by DARTS[24] and FB-NET[25] implementations that have used NAS on the small-scale CIFAR-10[48] data set to develop models for the large-scale ImageNet[49] data set. We created the proxy data set by randomly sampling 10,000 images from the original CCPD data set, which has over 200,000 LP images. Then we performed the NAS as described in Section 3.3, on this proxy data set to obtain neural network architectures.

However, this data proxy optimization problem has a constraint concerning the relative accuracy of the proxy and the original data set. The proxy data set can be biased and may not fully capture the entire original data set, Thus, the related accuracy obtained for the proxy data set may be less than the original data set. Therefore, after training the candidate architectures on the proxy data set, we performed a manual optimization on the original CCPD data set as well. This optimization process considers the probability distribution received of the NAS. Here, we use the argmax to identify the architecture instance with the maximum predicted probability for the proxy data set. However, this architecture instance may not give the optimal model for the original data set, due to the biasness of the proxy data set. Also, there can be an instance with a slightly lower probability, that gives better performance, if applied to the original data set. We use manual optimization to identify the anticipated instance among the instances close to the maximum probability, which can give a higher performance for the original data set.

Therefore, to find the optimal architectures for the larger data set, we retrained the resultant network on the entire CCPD data set (70% training, 15% testing, and 15% validation) and measured its performance. Then we used the architecture weights as a guide for the suitability

of each operation and tested slight mutations of the resultant network to see if those mutations will result in better performance in the larger data set. Therefore, the manual optimization process trains the original CCPD data set using the architecture that has shown a slightly lower probability in the proxy data set, and if there is an increase in the performance, then we fixed the final model with that instance. The best models selected by this process are described in Section 4. This identified optimal model will be used for the automated licence plate recognition process in the real environment. Thus, for each resultant model that we obtained from the NAS process, we performed this one-time optimization on the original CCPD data set and these final optimized models are proposed as the lite-LPNet for the process automation.

4 | LITE-LPNET ARCHITECTURES

4.1 | Overview of Lite-LPNet models

In this section, we introduce a family of optimal learning models called Lite-LPNet for license plate detection and recognition processes that execute purely on edge devices. In this study, we follow a two-stage approach for license plate recognition. As a result, there are two distinct sets of models in the Lite LP-Net family as shown in Figure 5.

One set is used for license plate detection (Stage 1) and the other for license plate recognition (Stage 2). Given the diversity of hardware capabilities in edge devices, we have considered three hardware categories namely low, mid, and high tier as follows:

1. Low tier (Raspberry pi zero),
2. Mid-tier (Raspberry pi 3 b+), and
3. High-tier (Raspberry pi 3 b+ & NCS 2).

Low-tier represents extremely power-efficient and cheap edge devices and we have used a Raspberry pi zero to represent the hardware of this category. Mid-tier represents hardware that requires the typical computational capabilities of an edge device. This tier represents a modern single-board computer with an ARM processor, without relying on dedicated accelerators, which makes them more computationally powerful compared with low tier devices but still cheap enough for mass deployment. For this, we have used a Raspberry pi 3b+ as the representative device. For the high-tier, we have considered the power-efficient edge devices with

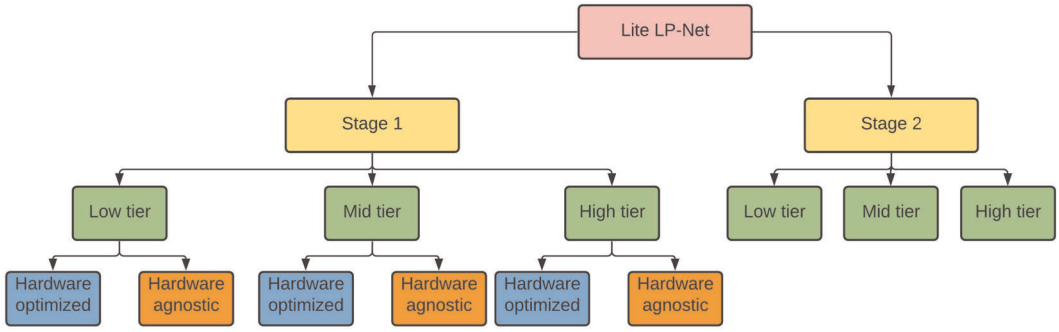

FIGURE 5 LITE-LPNet family [Color figure can be viewed at wileyonlinelibrary.com]

dedicated accelerators. This represents scenarios where the unit cost is not necessarily an issue, given that it is power efficient to be deployed in the field. We have used an Intel Neural Compute Stick 2 with a raspberry pi 3b+ as the representative device for the high-tier.

In Stage 1, LP detection process, we propose two models for each hardware tier. One of the models is optimized for the target hardware platform using FBNet[25] algorithm and the other model which is hardware agnostic using PC-DARTS.[24] Here, the Stage 1 models process every single image to detect whether there is a licence plate or not. On the other hand, Stage 2 models handle only the images that are already detected as license plate from Stage 1. Therefore, the latency of the detection models is critical compared to the processing of the recognition models. Since it is important to reduce the computational time in Stage 1 than Stage 2. Also, since the accuracy of detecting a licence plate is critical, we use both hardware optimized and hardware agnostic models for the detection process, and we describe the results in Table 2.

In contrast, Stage 2 computes the recognition task from the already detected license plates. Therefore, we use only hardware agnostic models intending to reduce the associated computational costs of executing the NAS at the edge devices. Further, having hardware optimized models in the recognition process will lead to extra cost, which can be challenging to use in the targeted practical applications. Therefore, Stage 2 that responsible for the LP recognition process, proposes three models, one per each tier using the PC-DARTS algorithm which is hardware independent. For the evaluation, we have used the Chinese License Plates Data set (CCPD),[16] which contains only seven characters in a license plate. Thus, the proposed model is developed for a specification with seven characters and are structured such that they can be modified to handle an arbitrary number of characters.

We have represented the neural network architectures using a set of diagrams for a detailed description. The naming convention is the same as in tensorflow.keras.layers API and the default parameters are the same as in TensorFlow version 2.3.0. To achieve concision, we have combined several repeated patterns into layer "abbreviations." The sequence of layers each of these abbreviations represents and how those layers are connected is shown in Figure 6.

The naming convention of the models in the Lite-LPNet family, we have followed the following convention and is provided in Table 1. The first two letters, s1 or s2 will represent whether it is a detection or recognition model. This will then be followed by a single letter l, m, or h representing if it is a low, mid, or high tier model. For a detection model, the letter h will be added if it is hardware optimized. As an example, "s1_m_h" is a mid-tier hardware optimized detection model.

4.2 | Lite-LPNet detection models: Stage 1

Detection models take the image from a camera and predict the bounding boxes of the license plate. We propose six different models for this task, and a given model is optimized either to reduce the latency or increase the accuracy. The low latency solution is recommended for our hardware specification with edge devices. The hardware optimized models are the results of the NAS using the FB-Net algorithm and here, we have used latency values calculated using the hardware setups for each tier to build three latency tables and, performed three separate architecture searches on each of them. The resultant network that is built using the architecture search results on each of the hardware tier's latency table is selected as the hardware optimized model for the corresponding tier, as shown in Figure 7.

Software configuration used for measuring those latency values is as follows. All Raspberry pi devices were running Raspberry Pi OS (32-bit) version August 2020 with desktop and

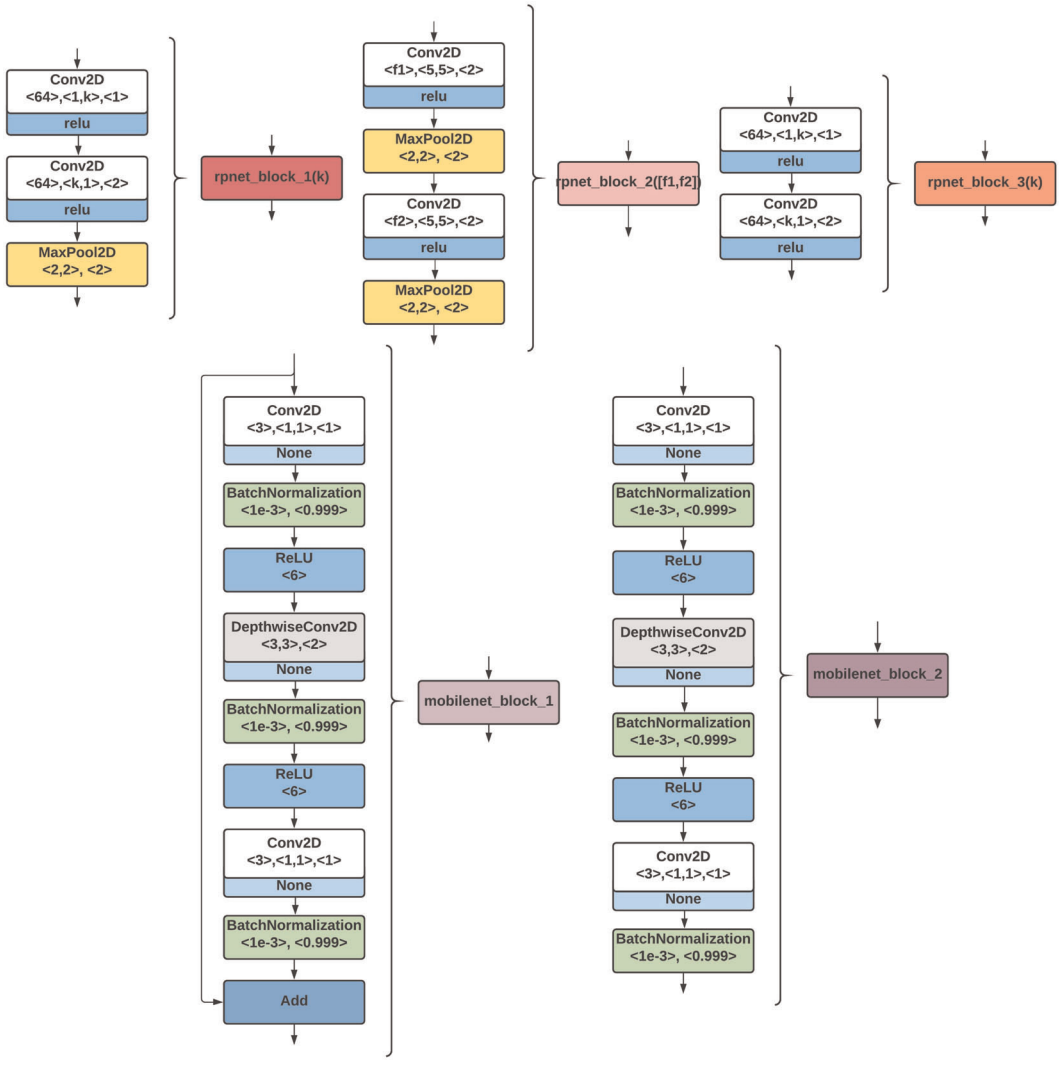

FIGURE 6 Layer abbreviations [Color figure can be viewed at wileyonlinelibrary.com]

recommended software. We used Tensor-Flow lite version 2.1.0 and Python 3.7.3 as the runtime on those devices. We used OpenVINO version 2019.3.376 to convert the Tensor-Flow models that were compiled using Tensor-Flow version 2.2 into the intermediate representations for the Intel Neural Compute Stick.

TABLE 1 Naming convention for the detection and recognition models of the Lite-LPNet family

	Stage 1		Stage 2
Hardware tier	**Hardware optimized**	**Hardware agnostic**	**Subnetwork**
Low-tier	s1_l_h	s1_l	s2_l
Mid-tier	s1_m_h	s1_m	s2_m
High-tier	s1_h_h	s1_h	s2_h

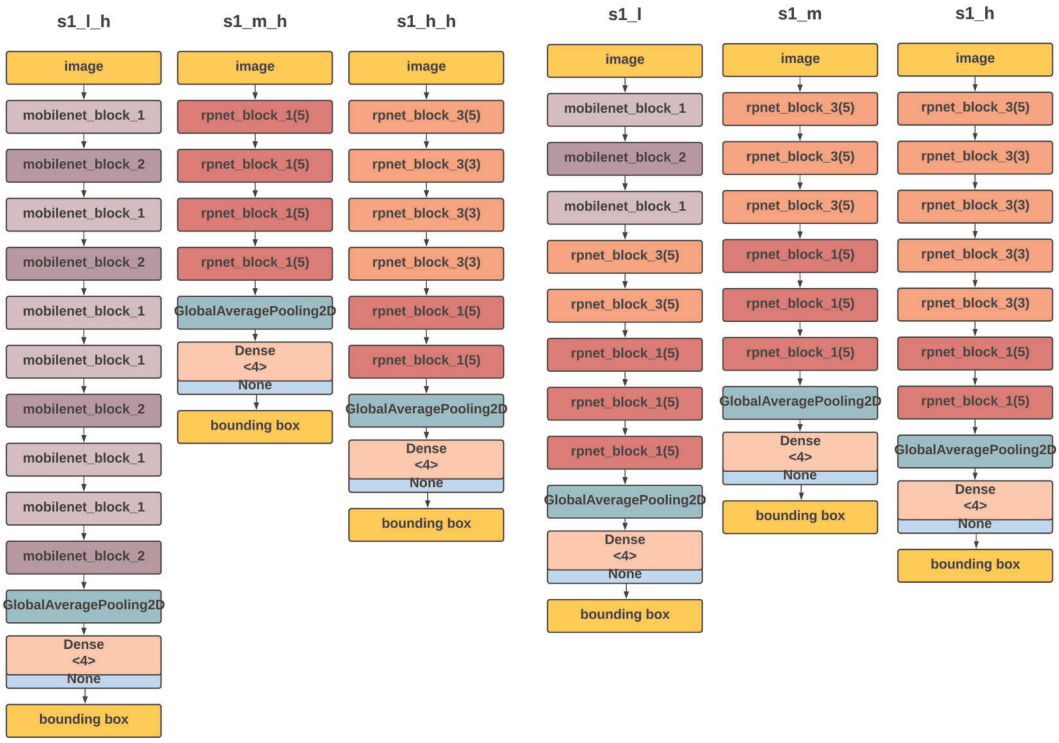

FIGURE 7 Detection models—Hardware optimized (left) and detection models—hardware agnostic (right) [Color figure can be viewed at wileyonlinelibrary.com]

The hardware-optimized models give low latency in the processing units which were used to build the latency table. However, they can give a subpar performance in similar but not identical processing units. This is because model optimization by its nature tries to exploit various nuances of the target processing unit. This limitation can be an issue when deploying models in production. For instance, instead of using a Raspberry pi 3b+, a developer may need to use a similarly priced and capable single board computer such as the Banana pi F2P which has a different processor. Although both of these single board computers satisfy the definition that we have used for mid-tier hardware set up, due to that subtle difference in the processing unit they may show significant different latency figures. To handle this unknown variability, we have presented another set of models that are not hardware optimized.

These models are based on the PC-DARTS algorithm and optimized to increase detection accuracy without regard to processing latency. In this case, we used the memory capacity of each tier's hardware setup as a fixed upper bound for the model's memory requirement. As a result, for instance, the model recommended for the mid-tier executes within the available RAM of a Raspberry pi 3b+ or any other board such as BPI-F2P which has a similar memory capacity. Since these models are not optimized to exploit nuances in the processing unit, latency figures should show less variation when the processor is of a similar design.

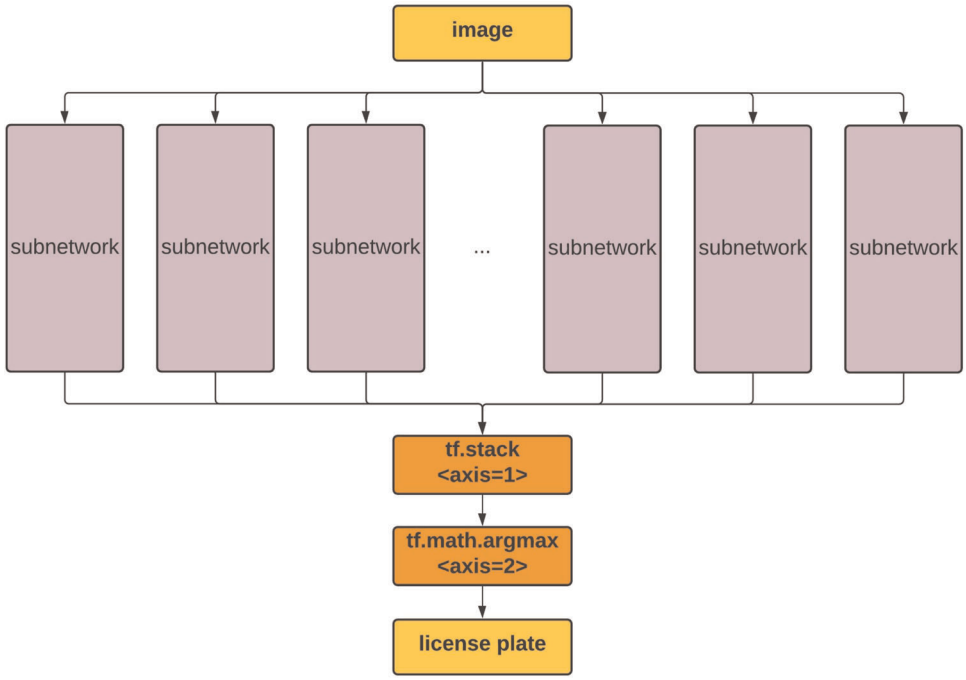

FIGURE 8 Recognition model structure [Color figure can be viewed at wileyonlinelibrary.com]

4.3 | Lite LPNet recognition models—Stage 2

LP recognition models take the cropped image of the license plate as the input and return a sequence representing the content of the license plate. We propose three hardware-agnostic models for LP recognition, by following the same process as used for the hardware-agnostic detection models. As described in Section 3.3, here we considered two different design paradigms: a design similar to the TE2E model and a design based on the RPNet model.

First, we tried to get two models (each following one of the two paradigms) for each hardware tier using the PC-DARTS algorithm. Then we picked the most accurate model out of the two as the model specific to a given hardware tier. Based on our testing results, the subnetwork based approach consistently outperformed the single model approach. As a result, the selected best three models have followed the RPNet based second approach and Table 1 gives the model naming conventions for the recognition models. Figure 8 shows the proposed high-level model structure for the license plate recognition with n characters. In the existing RPNet model, all the subnetworks are trained at the same time. However, our model trains each subnetwork separately that give two main advantages: (1) Individual training of subnetworks allows us to fine-tune models for each character separately and (2) All three proposed networks differ only in their subnetwork.

The architectures of the three recognition subnetwork are shown in Figure 9. It is assumed that each character can take n possible values. The entire set of characters in the license plate is the input for each subnetwork. Finally, the consecutive output values of each subnetwork form the recognized license plate number.

Based on the existing constraints, several aspects were considered to calculate the memory requirement for a model. First, We have defined this model for the structure of a valid license plate

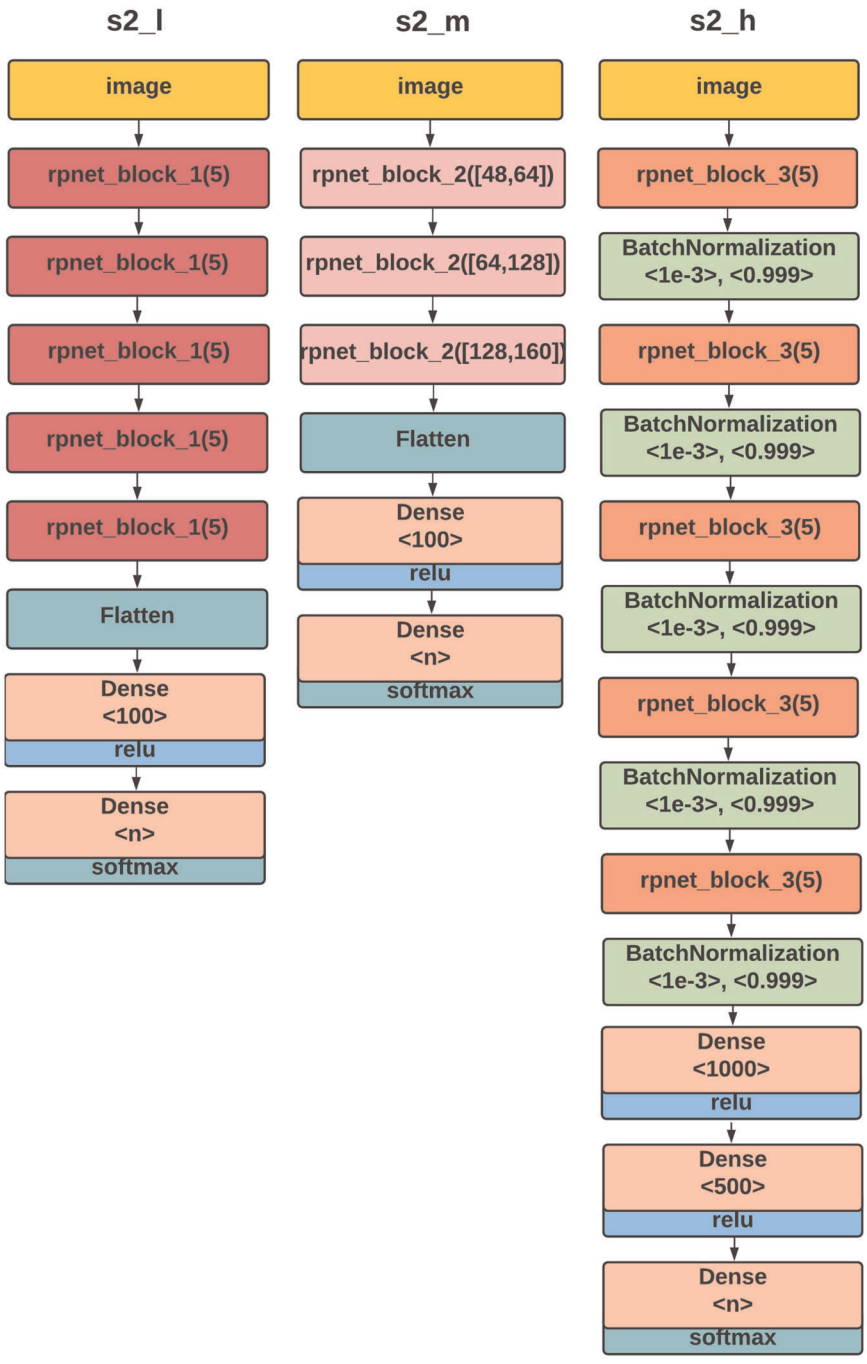

FIGURE 9 Recognition subnetworks [Color figure can be viewed at wileyonlinelibrary.com]

in China, using the CCPD data set.[16] Therefore, there are seven characters on the license plates. For each character in the LP, there are 35 possible values. The memory requirement of each model is calculated using seven identical subnetworks. For example, for the low-tier configuration, we can use seven s2_l models to build the final model. We can also define the memory limit by

considering the available memory of the low tier hardware setup as the upper bound for the memory requirement for the PC-DARTS-based algorithm to search for the optimum architecture.

At the production level, it is possible to mix these subnetworks if the device has sufficient memory. For instance, for a low-tier configuration, we can use six s2_l submodels and one s2_h submodel to build a new recognition model. Here, no additional training is required as all the learnable parameters are in the submodels. This set up can be used when a given one character is more difficult to detect compared to the other six characters. This helps to increase the recognition accuracy of that character by using a higher tier model to identify that relatively complex character. Although this setup requires more memory than using seven lower-tier models, it will still need less memory than using seven higher tier models. Moreover, since the subnetwork consists of the learnable parameters, the model is not needed to retrain after swapping the subnetworks. We call this process submodel mixing. Additionally, it is observed that the models recommended for the license plate recognition in Stage 2 requires more memory consumption than the detection process in Stage 1.

4.4 | Lite-LPNet pipeline implementation

One of the main advantages of using a two-stage process over a single-stage model is the reduction of effective memory usage. A single-stage model will have less total memory consumption compared to the total memory consumption of the two models used for each stage in the two-stage process. This is due to the parameter sharing in a single-stage model. However, the memory consumption of each stage's model in a two-stage process will be less than the single-stage model. This is because the single-stage model can both detect and recognize the license plate unlike two-stage models, which performs one of the processes. Therefore, the two-stage model can have less effective memory consumption than a single-stage model, by loading only one of the two models into the memory at a given time.

Moreover, a complex model can have better accuracy than a less complex model. Thus, there is a potential to engineer more accurate models for each stage due to the higher memory availability in the two-stage approach. This also simplifies the training process compared to the end-to-end training required for a single-stage model. Since the memory capacity is a major limitation in edge devices, the two-stage process has a chance to achieve high accuracy while keeping the model sizes small enough to fit in the RAM of the edge device. Furthermore, the hardware platforms of the edge devices lack the processing capacity to execute multiple subnetworks in parallel. Thus, even both detection and recognition models are in memory, only one model executes at a given time, and the other model keeps idle and consumes memory.

Therefore, by having only one model in memory at a given time, it is possible to execute a larger but more accurate model without consuming memory on a second model. This can be considered as a best design practice, as free memory does not give any significant benefit to the system. Thus, maximizing memory utilization to get better accuracy is a good trade-off. However, this trade-off comes at a cost of latency, as it consumes time to remove the already processed single stage-model and load the Stage 2 model to the device RAM.

However, the process of keeping only one model in memory at a time can have a caveat. For instance, the processing a single image involves several stages as follows:

- Task 1: load the detection model from the persistent storage into the device memory and initialization.
- Task 2: pass the camera image through the detection model to obtain bounding boxes of the license plate.
- Task 3: crop a license plate.
- Task 4: garbage collection of the detection model.
- Task 5: load the recognition model from the persistent storage into the device memory and initialization.
- Task 6: pass the cropped image through the recognition model to obtain the content of the license plate.
- Task 7: garbage collection of the recognition model.

Here, Task 1 and Task 5 are input–output operations that can take a long time depending on the capabilities of the persistent storage used. Additionally, it involves two garbage collection pauses to process a single image (Task 4 and Task 7). These two factors can negatively affect the perceived processing latency of the pipeline. Therefore, we make two suggestions called batch processing and model downgrading to alleviate the processing latency of the pipeline.

Instead of processing each image from end-to-end, use a batch of images for every image that triggers a complete cycle of the processing pipeline. Thus, each of these batches of images can process using the proposed pipeline. Since several images are loaded to a buffer and fed to each stage as a batch, it reduces the latency. This will reduce the amortized processing cost because those input–output operations (Task 1 and Task 5) and garbage collection steps (Task 4 and Task 7) happen only once per batch, hence reducing the time taken to load and offload models per each image separately. Thus, for large batch sizes the amortized time for each image is low, while the memory consumption is high. However, to achieve this, the images from the camera need to be buffered, which cannot be used for a real-time application. Thus, there are issues in performing batch processing.

The model recommendation (Sections 4.2 and 4.3) has assumed that only one model will be in memory at a given time. Therefore, the recommended models may not be suitable when both models are needed to keep in memory. For instance, the memory capacity of the mid-tier device may not be sufficient to have both detection and recognition models in memory at the same time. However, in such cases, the models recommended for the lower-tier platforms that require less memory can be used in mid-tier devices. This is referred to as model downgrading and Section 4.5 provides guidelines for selecting these models. By having both detection and recognition models in memory simultaneously, Task 1 and Task 5 need to perform only once as long as the device is powered on and garbage collection pauses can be ignored. Consequently, by model downgrading, the images can be processed in real-time without buffering in batch processing.

4.5 | Model downgrading approach

4.5.1 | Input size-reduction

We suggest reducing the input size of the models before downgrading models from a higher tier to a lower tier. In this study, we have defined an input size specification for model

recommendations. Here, the size of the input for Stage 1 detection process is considered as 480×480 pixels and for Stage 2 recognition process is considered as 280×560 pixels. The reduction of data available for the model results in having few parameters thus requires less memory consumption. However, this data reduction can decrease accuracy and requires model retraining. Better detection and recognition accuracy can be obtained by changing models to use high-resolution images, which requires more memory capacity.

4.5.2 | Model downgrading

We suggest model downgrading if the input size reduction is not used for a given model. Instead of directly downgrading both the detection and recognition models, we suggest a granular approach with a mix-and-match method, based on the following observations:

1. The recognition models have significantly higher memory consumption compared with the detection models
2. The difference in memory consumption among hardware tiers is most pronounced in recognition models
3. If the detection stage fails to localize the license plate accurately, then the recognition stage will invariably fail as it requires the localized license plate as an input.

Therefore, it is suggested to use the recommended detection models in this solution. For instance, the mid-tier hardware configurations may perform well with the detection models that are optimized for the mid-tier platforms. Then by selecting a lower-tier recognition model for Stage 2, the process requires less overall memory in contrast to using mid-tier models in both stages, detection and recognition processes. There is a considerable difference between memory consumption among the models in the two stages. For instance, Stage 2: recognition models require high memory consumption, compared to models in Stage 1: detection. Thus, having a high-end model at Stage 1 and low-end model at Stage 2 will reduce the total memory requirement. This setup will consume less memory than using high-tier models for both stages and give better accuracy than using low-tier models.

4.6 | Lite LP-Net with limited data

The main limitation to train deep neural networks for the task of recognizing license plates is the lack of large, annotated data sets. Although several such data sets exist such as CCPD,[16] they are not generic data sets as they depend on a specific region or country. To circumvent this issue, we propose two solutions namely (1) transfer learning on license plate detection and (2) synthetic data generation for license plate recognition.

4.6.1 | Transfer learning

Input for the license plate detection model is diverse in nature. Although the proposed research scope is limited to the inputs with only one license plate in an image, based on the factors such as the type of the vehicle, distance to vehicle and camera angle, there is a large variation in the

location of the license plate within the entire image. In addition, the detection model must be robust against the background noise-related factors such as text appearing in the images in places other than the license plate. Thus, Stage 2: recognition process highly depends on training on a diverse data set. The factors such as the location of the license plate, general shape of the license plate remain constant across regions. Thus, the license plate detection process is less dependent on the region compared with the license plate recognition process. Based on that observation we propose the following fine tuning approach to do transfer learning on our stage-1 models.

First, train the model using a large public data set such as CCPD[16] until convergence. Then freeze all the layers except the dense layers (last prediction layer) and train the model on the region-specific data set until convergence. Then unfreeze the entire network and retrain the model with the region-specific data set but with a smaller learning rate. The above model fine-tuning process utilizes the large public data sets to train license plate recognition models for regions, where such a data set cannot be economically collected.

4.6.2 | Synthetic data generation

A large and diverse data set supports to train a learning model robustly. However, there is a lack of large-scale nighttime license plate image data sets that can be used for processing. The input for the recognition process shows less variation compared to the detection stage as the license plates are standardized. We used a synthetic image generation technique to convert the CCPD data set to nighttime images. Therefore, a synthetic data generation process can be used to produces all the possible license plates from a small data set for the recognition stage using the following approach.

First, we gathered a sample set of images, that contain all the valid characters that can appear on a given license plate. These images were cropped to extract the characters from the images and annotated the areas of each character in several plates. The image segmentation technique such as Mask-RCNN[50] can be used for this process in a semisupervised manner. Then, we replaced the areas annotated with cropped-out characters by swapping characters to create new images as shown in Figure 10. Finally, we inferred the content of the new image, completed its annotations, and trained the Stage 2 models on this annotated data set.

Although this process adds slight changes to the background of the image around the characters, the proposed models and training process have shown sufficient robustness against this effect. Thus, Lite LP-Net modes can be used with small data sets as well by fine-tuning the detection model and generating synthetic data for the recognition model.

FIGURE 10 Synthetic license plate generation [Color figure can be viewed at wileyonlinelibrary.com]

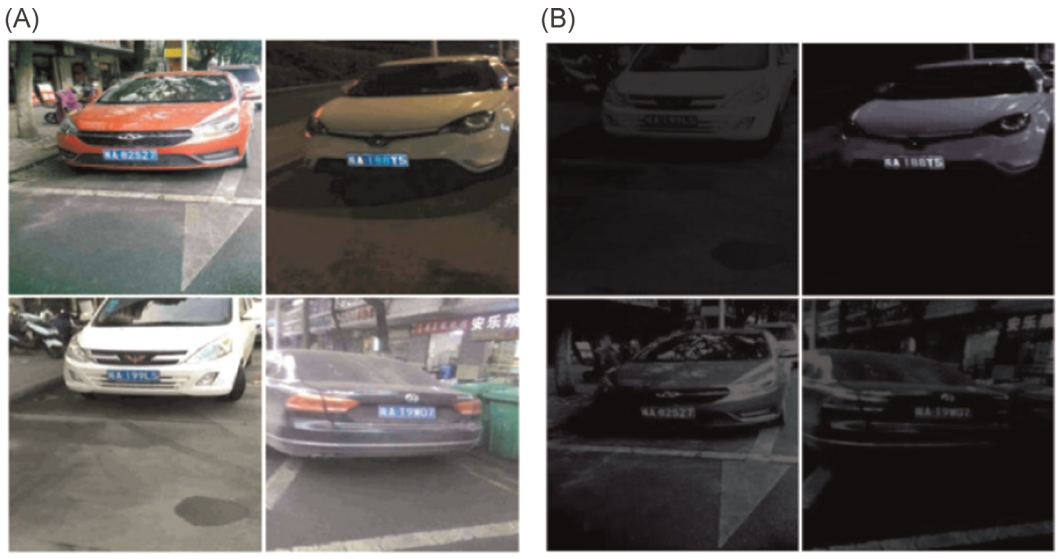

FIGURE 11 Sample CCPD[16] images (A) daytime and (B) generated synthetic nighttime images [Color figure can be viewed at wileyonlinelibrary.com]

5 | RESULT ANALYSIS

5.1 | Experimental setup

The proposed solution has been evaluated to measure the performance of the neural network models with diverse license plate data sets containing daytime, real and synthetic nighttime images. Here, we considered several evaluation metrics such as average precision for detection, accuracy for recognition, latency and model size.

5.1.1 | Data set

Although there exist LP data sets like the Chinese City Parking Data set (CCPD),[16] to achieve different variations of license plates, they have mainly focused on daytime images. Curating such a data set for nighttime images is both expensive and time consuming. To mitigate this issue, we used a synthetic image generation technique to convert the daytime RGB images of the CCPD data set to thermal infrared (TIR) nighttime images.

Accordingly, we used CCPD with 200,000 images to evaluate the system with daytime and synthetically generated nighttime license plate images. We have used fivefold cross validation as describe in Section 5.1.2, where each fold consists of 40,000 images. In addition, the proposed model is tested in real environment with 100 nighttime images.

The TIR image generation for license plate detection was synthesized using the method proposed by Zhang et al.[51] Here, we used a GAN based pix2pix model for image translation and it was trained using the biggest available multispectral data set named KAIST,[52] which has a significant amount of RGB and TIR images. We trained the pix2pix network from scratch and the weights were initialized from a Gaussian distribution with a mean 0 and *SD* of 0.02.

We enlarged the input images to 480×480 and trained the pix2pix network for 100 epochs with a decaying learning rate of 0.0002, lambda_l1 of 120.0 and keeping other parameters the same as the original pix2pix[53] paper. Figure 11 shows some sample daytime images from the CCPD along with their corresponding synthetically generated TIR images.

To generate nighttime data for training the recognition model, we converted the RGB images of CCPD data set to grayscale using matplotlib and set the color map to grey. The followed methodology is described in Section 4.6.2. The presented method preserves the image quality and avoids generating incomplete license plate characters that are difficult to read.

For a complete validation of the proposed methodology, we collected a real-world data set. For this, we deployed our prototype under operational conditions configured to save input from the camera as a video to the internal storage. We ran this process for 7 days between 8 p.m. and 4 a.m. Then we went through the recorded video and sampled frames with unique vehicles. We curated 100 images which were then manually annotated. We used the transfer learning technique to train models for Sri Lankan license plates and then validated the performance of our models.

5.1.2 | Evaluation metrics

We have measured the correctness of the models in terms of their ability to locate the license plate in the detection process (Stage 1) and ability to correctly recognize the license plate in the recognition process (Stage 2). Since Stage 1 models predict four continuous values representing the bounding box, we have used root mean squared error (RMSE) as the loss function for the Stage 1 models. However, these metrics are too general to properly reflect the specific nature of the bounding box prediction problem. There is no objectively correct bounding box as suggested by regression matrices. Instead we require a bounding box that covers the license plate while having as little area as possible. Generally, average precision (AP) at a fixed intersection over union (IOU) threshold is a metric used for object detection.[54] Therefore we have used AP and mean IOU as the evaluation metrics for the Stage 1 models. For Stage 2 models we consider prediction to be accurate if and only if every single character in the license plate is recognized correctly.

Considering the evaluation metrics, the cross-validation approach evaluates the data set by repeatedly split into a training and a validation data set. Therefore, cross-validation performs better for unseen data, than residuals that use the enter data set. The holdout method, which is a simplest cross-validation method, splits the data set into only training and testing set, and may have a high variance in the evaluation based on the split ratio. The K-fold cross-validation splits the data set into k subsets and repeats the holdout method for k times.[55] In each iteration, the k-fold cross-validation method uses $k-1$ parts for training and one part for testing. Since the process repeats for k time, each part gets the chance to appear in the test set during the k iterations. Consequently, considering the execution of k iterations, the total of k parts will be tested at the end. Then, it calculates the average error across all k sets. Here, the resultant prediction variance becomes less with the increase of the number of sets that the entire data set is divided. However, the increase in the number of folds will consume more time.

Here, when the number of folds decreases, the accuracy becomes low. When the number of folds increases, the bias in results decreases. Moreover, when there are many folds, the number of times needed to train the model and the amount of data used for training increase, hence, the computational cost will be increased. Also, when the test fold is small the variance will be

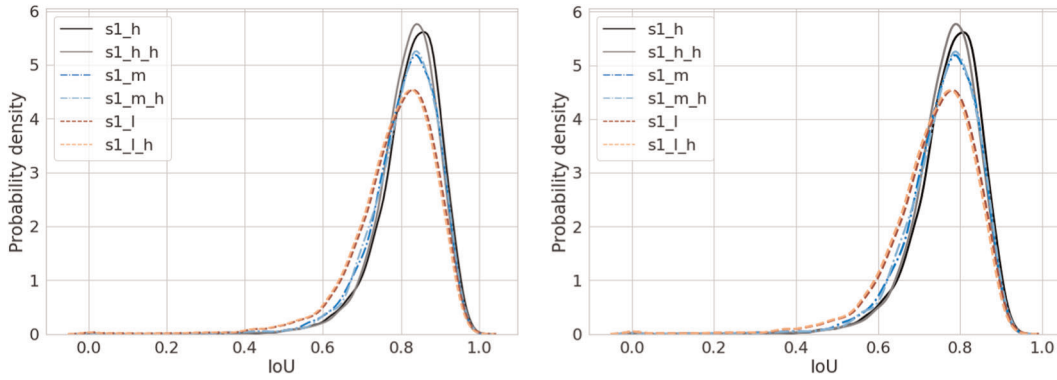

FIGURE 12 Distribution of intersection over union values for detection models, (left): day time; (right): night time [Color figure can be viewed at wileyonlinelibrary.com]

increased. To maintain the trade-off between the accuracy and the computational cost in the edge device, we have used fivefold cross-validation as a middle ground. Thus, less dependent on the method of data splitting, and each set may have the same data distribution.

We divided the CCPD data set of 200,000 images into five equally sized random partitions, where each fold consists of 40,000 images. Then we evaluated the performance for each partition and calculated the average value for each metric. Also, we ensured the images from each partition in daytime images are used to generate images in the respective nighttime partition, to evaluate the performance of the data sets. Here, every data point belongs to a test set once and be in a training set four times. Cross-validation allows detecting differences in results sets and reduces the impact by taking the average, as described above. Accordingly, the fivefold cross-validation improves the results with the best parameters, avoids overfitting and assess the generalization of the results of statistical analysis to an independent data set.

TABLE 2 Performance of Stage 1 license plate detection models

	Daytime		Nighttime			Latency (s)		
				Average precision				
Model	Mean IOU	Average precision	Mean IOU	Synthetic	Real	Server	Target hardware	Size (MB)
s1_h	0.8226	0.9284	0.7726	0.8451	0.85	0.08667	0.01187	0.77762
s1_h_h	0.8163	0.9299	0.7664	0.8401	0.9	0.00016	0.01143	0.87075
s1_m	0.8101	0.9005	0.7601	0.7982	0.85	0.01435	0.15698	0.68690
s1_m_h	0.8107	0.9029	0.7607	0.7962	1.0	0.00035	0.00417	0.68307
s1_l	0.7885	0.8422	0.7386	0.7146	0.95	0.01396	4.54	0.55685
s1_l_h	0.7835	0.8327	0.7336	0.6987	0.95	0.00093	4.08	0.56253
RPNet	_	0.9450	_	0.8460	0.8589	0.01539	_	210
TE2E	_	0.9420	_	0.8352	_	0.21339	_	_
Yolo-V3	_	0.8723	_	0.7086	0.728	_	_	227

Abbreviation: IOU, intersection over union.

5.2 | Result analysis of license plate detection and recognition

5.2.1 | Experiment with daytime images

We have used CCPD data set with 200,000 images for the evaluation of the system with daytime images. Figure 12 (left) shows the distribution of IOU values for all the detection models, where the line graphs show the kernel density estimation using a Gaussian kernel. Generally, a model that performs well have a higher mean IOU value as it indicates better performance and a low *SD*. A higher *SD* indicates the model's performance is more likely to change depending on the input data. Therefore a good model's IOU distribution is a right-skewed narrow distribution. As shown in Figure 12 (left), the Stage 1 models in the detection process satisfy this requirement. Here, the AP follows the same trends as the IOU with both higher tier models performing better than lower-tier models and as well as models of the same tier showing similar performance.

Table 2 show the resulting average precision, latency, and memory consumption of the proposed detection models executed on Raspberry pi 3b+ and Intel Neural Compute Stick 2. Further, for comparison, we have included corresponding results obtained by the existing state-of-the-art algorithms namely RPNet,[16] TE2E,[17] and Yolo-V3.[15] However, these existing models were tested with server-grade hardware, as they cannot run on the edge-devices addressed in this study. The naming conversions of the proposed models are stated in Table 1, in such a way that s1_h_h and s1_h represent the high-tier hardware optimized and agnostic models, respectively. Here, we have shown the results obtained for the three hardware platforms low tier (Raspberry pi zero), middle tier (Raspberry pi 3b+) and high tier (Raspberry pi 3b+ with Intel Neural Compute Stick 2). The model names starting with s1, and s2 represents the proposed models for the LP detection stage and recognition stage, respectively.

In this study, we aim to execute deep learning models in edge devices. Therefore, the main consideration is the memory requirement of the hardware. In this scenario, we do not consider latency reduction as a limiting factor. Thus, the best contribution of this approach is the efficiency considering the performance of the given model size. Thus, our focus was to obtain models that are competitive with state-of-the-art models for servers, while being resource-efficient to run on edge devices. As stated in Table 2, Stage 1 hardware optimized models perform slightly faster than hardware-agnostic models, except for the mid-tier solutions that have a considerable difference in the latency.

Moreover, we have measured the latency of the proposed detection models in both the target edge devices that consists of Raspberry pi 3b+ and Intel Neural Compute Stick 2, as well

TABLE 3 Performance of Stage 2 license plate recognition models in terms of accuracy and efficiency

		Accuracy (nighttime)		Latency (s)		
Model	**Accuracy (daytime)**	**Synthetic**	**Real**	**Server**	**Target hardware**	**Size (MB)**
s2_h	0.9987	0.9476	0.9873	0.01322	0.02176	183
s2_m	0.9877	0.9382	0.9882	0.01255	0.14839	11.7
s2_l	0.9565	0.9054	0.9586	0.00514	6.2	4.5
RPNet	0.9876	0.9736	0.9895	0.01539	_	210
TE2E	0.9789	0.9437	0.9569	0.21339	_	_

as in a server environment for the testing purpose. For the server environment we have used an n1-standard-4 virtual machine from Google Cloud Platform equipped with an Nvidia T4 GPU. We measured the latency for a batch of 36 images and measured the average time to process a single image.

In Table 3, we have shown the accuracy of the license plate recognition stage models and we see the same trends that we have observed for the detection process. Here, the proposed models are hardware agnostic and the higher tier models perform better than lower-tier models.

5.2.2 | Experiment with nighttime images

For night time evaluation we used two data sets: a generated synthetic nighttime data set and a real nighttime data set. Figure 12 (right) shows the IOU distribution we obtained for the synthetically generated data set and it is almost similar to the daytime distribution in the left. This indicates that the models behavior is same for both day and night time images. However, we also observed that curves have shifted slightly to the left for nighttime data compared to the daytime. Therefore, models will have more difficulty detecting license plates at night. Thus, the accuracy values can be slightly higher for the daytime images. The evaluation results of the nighttime images for the detection and recognition processes are given in Tables 2 and 3, in respective columns.

Moreover, we have evaluated the models on a real-world nighttime data sets. First, we trained our models on the synthetic data set and then fine tuned them on an annotated data set as described in Section 4.6 and tested the resulting models on 100 images. The results of this experiment are also shown in Table 2. Similar to our analysis of detection models, the relative performance of the Stage 2 recognition models has remained the same for both day and the generated night data sets. However, daytime data shows more accuracy than the nighttime data, in both processes.

5.3 | Efficiency of lite LP-Net models

We used two matrices to measure how efficiently the model utilizes available hardware resources called latency and model size. We considered the latency values in both server-grade and target hardware with Raspberry pi 3b+ and Intel Neural Compute Stick 2. The latency in a server environment is measured with an n1-standard-4 virtual machine from the Google Cloud Platform equipped with an NVIDIA T4 GPU. We used a batch of 36 images and measured the average time to process a single image. Then, we have measured the actual model latency in the target hardware platform. For this, we have run each model in their respective edge device and measured the latency to process a single image. The results of this experiment are shown in Tables 2 and 3. To remove the effect of external factors such as operating system scheduling, we have run each experiment 1000 times and taken the average reading in both cases. For the experiment with server-grade hardware, we used Tensorflow version 2.3.0 and CUDA version 10.1 with NVIDIA driver version 418.6. Latency value does not include the time it takes to load data from disk to system memory. However, it does include the time taken to transfer data from system memory to GPU as well as the time it takes to send results back from the GPU in addition to the inference time.

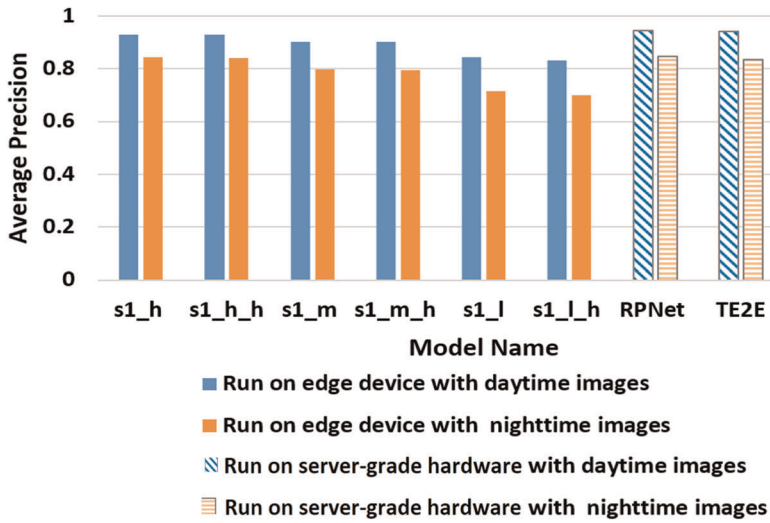

FIGURE 13 Comparison of average precision of each detection model for daytime and nighttime [Color figure can be viewed at wileyonlinelibrary.com]

The hardware configuration of the actual edge devices is as follows. All Raspberry pi devices were running Raspberry Pi OS (32-bit) version August 2020 with desktop and recommended software. We used Tensor Flow lite version 2.1.0 and Python 3.7.3 as the run-time in those devices. We used Open-VINO version 2019.3.376 to convert Tensor-Flow models that were compiled using Tensor-Flow version 2.2 into intermediate representations for the Intel Neural Compute Stick. Tables 2 and 3 shows the average latency of processing a single image in these hardware platforms. As expected, we can see that hardware optimized models have lesser latency than the hardware-agnostic models. To measure the model size, we have used the memory usage of each model's Tensorflow flat buffer's size as the model size and the results are shown in Tables 2 and 3. While the actual amount of memory allocated by the Tensorflow Lite interpreter to run the model is different from this exact value there is no recommended or obvious way to measure that memory allocation. This value gives a lower bound for that memory allocation.

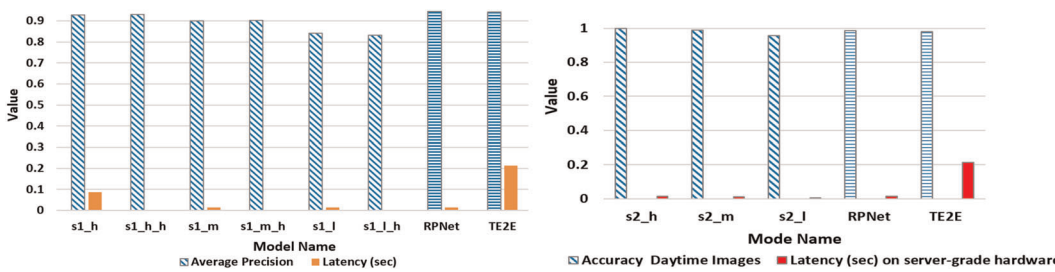

FIGURE 14 Latency versus accuracy in the server environment, (left): detection models and (right): Rrecognition models [Color figure can be viewed at wileyonlinelibrary.com]

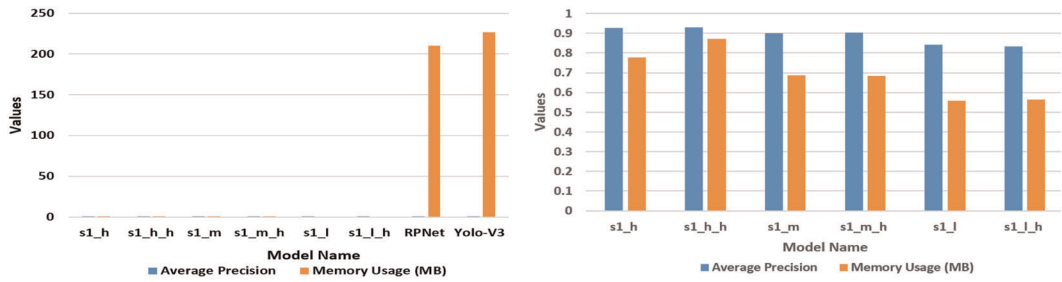

FIGURE 15 Detection model size vs average precision for daytime, (left): comparison with the existing models and (right): enlarged view of the model results [Color figure can be viewed at wileyonlinelibrary.com]

5.4 | Result comparison with the state-of-the-art models

In Table 2, we have shown the performance of the existing state-of-the-art models in ALPR and object detection domains, RPNet, TE2E, Yolo-V3, over the same data sets and the results are depicted in Figure 13. However, there were executed in server-grade hardware and computationally expensive to execute in edge devices as theses algorithm requires more memory requirements.

Since our models are designed for edge devices, the complexities of the proposed models are relatively low compared to server-grade models like RPNet and TE2E. Although, the accuracies of the proposed models do not outperform the existing algorithms, our aim of this study is to show the competing results of the proposed models that can be run on edge devices. At the same time, all models except the lower-tier models show superior performance to Yolo-V3,[15] which is a popular general-purpose object detector that has been used in several license plate detection research.[56,57] This shows that our models are competitive with the existing state-of-the-art solutions in terms of accuracy.

As shown in Table 3, the higher-tier models in the recognition process outperform the current state-of-the-art models such as RPNet,[16] in contrast to the results of the detection process, which is shown in Table 2. The comparison of daytime and nighttime precision values is shown in Figure 13. We can see that our higher-tier models have closed the gap between them and the current state-of-the-art RPNet[16] model. At the same time, the lower-tier models have closed the gap between them and Yolo-V3.[15] This indicates while detecting license plate is difficult in general for all models, the proposed models detect licence plates at nighttime relatively easier than the commonly used models. It indicates the better suitability

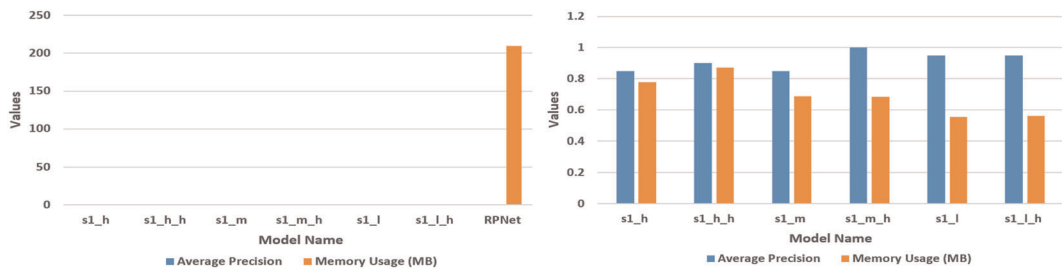

FIGURE 16 Detection model size vs average precision for nighttime, (left): comparison with the existing models and (right): enlarged view of the model results [Color figure can be viewed at wileyonlinelibrary.com]

of our models for nighttime operations. Figure 14 (left) and (right) compare the model latency with the average precision in a server environment for the CCPD original data set in daytime conditions for detection and recognition models. For a fair comparison, we have shown the obtained results for this server environment, as the existing models like RPNet are not implementable on edge devices. This clearly illustrates how our models show good average precision while maintaining significantly small latency which indicates models efficiency in terms of computing resource utilization. We can see that our Stage 2 recognition process models show competitive accuracy with the current state-of-the-art license plate recognition models while showing better model latency. This demonstrates our models are more efficient at resource utilization compared with the existing state-of-the-art models.

Moreover, Tables 2 and 3 show the size of the proposed models compared with the existing state-of-the-art models such as RPNet.[16] It must be noted that RPNet[16] is a single-stage model and therefore it contains both detection and recognition models in a single stage. For a comparable result with our models, one must add the size of Stage 1 and Stage 2 models. Yolo-V3[15] on the other hand can be used for both stages but in that case, the model size must be multiplied by 2.

Figure 15 compares the model size vs average precision of detection models for the CCPD original data set in daytime conditions. Similarly, Figure 16 shows the comparison of the model size versus average precision of detection models for the real night-time data set. In both Figure 15 (left) and Figure 16 (left), the memory consumption values of the proposed models are negligible in comparison to existing models. Therefore, the enlarged views are shown in Figure 15 (right) and Figure 16 (right), where the y axis shows the values correspond to average precision and model size. Also, these values are given in Table 2. Since the proposed models show very low memory consumption, they can be executed in the edge devices with low resources, with competing average precision levels. Thus, fulfill the objectives of this study. In addition, the results of the synthetic night time data are given in Table 2.

This demonstrates one of the major advantages of Lite LP-Net models. They show comparable performance to state-of-the-art models while being significantly smaller. This size advantage is crucial when it comes to running models in memory constraint edge devices. Thus, the results show the usefulness of the proposed automated licence plate detection and recognition approach implemented for edge devices.

6 | DISCUSSION

6.1 | Main contributions of the study

This study presented a novel approach based on NAS strategies for a hardware-efficient ALPR solution that runs purely on edge devices. The main focus of our study was to obtain models that are competitive with state-of-the-art models for servers, while being resource-efficient to run on edge devices. The system evaluation has shown an accuracy competitive to the state-of-the-art solution designed to run on server-grade hardware such as RPNet, which consumes more memory and computationally expensive to execute on edge devices.

This study has provided a basis for the research on nighttime license plate recognition by exploring synthetic data generation approaches to mitigate the issue with the scarcity of a large

TABLE 4 Comparison of existing license plate recognition studies

Related study	Data set	Resource requirement	Performance	
			Latency	Accuracy
Edge-based detection with Sobel filters[5]	2943 images	Intel Pentium 4 (2.4 GHz), 1 GB RAM	Yellow-color images: 92 ms and nonyellow color images: 46 ms	Yellow-color images: 95.55% and other images: 98.94%
Edge-based detection with Sobel filters[58]	610 gray scale images	Intel Pentium 3 (700MHz)	–	95%
Texture-based detection with Gabor Filter[2]	1024 × 768 pixel images	Xilinx Virtex IV	0.5 s	97.12%
Texture-based detection with Wavelet Transform[59]	1436 images	Intel Pentium 4 (1.6 GHz), 256 MB RAM	0.2–0.5 s	91.70%
Deep learning with General purpose object detectors[12]	SSIG and UFPR-ALPR	NVIDIA Titan XP	SSIG—Detection: 4.0654 ms, Recognition: 11.5164 ms, UFPR - Detection: 3.9292 ms, Recognition: 11.5591 ms	SSIG—Detection: 100%, Recognition: 97.83%, UFPR - Detection: 98.33%, Recognition: 90.37%
Deep learning with CNN model[60]	58950 license plate and 368000 character images	Intel Core i7 (2 GHz), 8 GB of RAM	–	*f*-score—Detection: 91.3%, Recognition 94.8%
Deep learning with CNN model[16]	CCPD	Intel Core i7 6700 (3.4 GHz), 24 GB of RAM, NVIDIA Quadro P4000	0.01639s	Average precision- Detection: 94.5%, Recognition: 95.5%
Proposed Lite-LPNet	CCPD 200000 images and 100 real night time images	Raspberry pi 3b+, Intel Neural Compute Stick 2	Detection (s1_h_h): 0.011s, Recognition (s2_h): 0.02176s	Detection (night time): 90%, Recognition (night time): 98.73%

and diverse nighttime license plate data set for training the learning models. Moreover, the transfer learning process helps to fine-tune the detection process by overcoming the issues related to limited large data sets.

Although neural networks for license plate recognition is a well-explored area for the daytime images with the server-grade hardware specification, we have provided the basis for ALPR with limited resources in constraint environments and complex parameters as future research. We have evaluated the models using 200,000 daytime license plate images from CCPD data set and the corresponding nighttime images generated synthetically. Also, the models are tested with 100 nighttime images captured in a real environment. Thus, the experiments can be easily extended for different other data sets. Therefore, the proposed solution balances the trade-off between the model size and the accuracy overcoming the challenges faced when developing an ALPR system for edge devices.

6.2 | Comparison of the proposed approach with existing studies

The early attempts to solve the ALPR problem were based on traditional computer vision techniques that are primarily focused on features of the license plate such as size, color and texture. Although these traditional approaches exhibit impressive performances, still most of these methods are tested on relatively small data sets. Therefore, the traditional computer vision-based system performances are less robust for image variations like license plate rotations, lighting conditions, and adverse weather conditions with large data sets. In contrast, the deep learning methods are trained and evaluated on larger and diverse data sets and, showed empirical results that are robust for image variations. Thus, deep learning approaches build accurate and powerful models against varying conditions. Table 4 summarize the comparison of related studies.

Moreover, for a reliable comparison of evaluation, we measured our performance against state-of-the-art deep-learning models in the ALPR context which are TE2E[17] and RPNet,[16] as stated in Tables 2 and 3 and analysed in Section 5.4. In general, the license plate detection process can also be generalized to a single class object detection task and the license plate recognition to an optical character recognition task. Therefore, we evaluated and compared our system with YOLO-v3,[15] which is currently one of the most applied models in object detection applications.

As shown in Figure 13, all the suggested mid-tier and high-tier models in Stage 1 for daylight, except the low-end configurations, have shown better performance compared to related work based on YOLOv3. However, the obtained AP values in daytime are less than the related work on RPNet. This is mainly because both YOLOv3 and RPNet modules are relatively large and complex when compared to all the models that were suggested in this current study. However, RPNet or Yolo-V3 based models do not execute on the defined low-tier and mid-tier hardware configuration, due to memory limitations. As a result, we can see that the proposed detection models are competitive with the high-end models designed to execute on server grade hardware, while at the same time being efficient enough to run on edge devices.

Despite the accuracy and latency, most of the existing deep learning solutions in ALPR have some problems in use with edge devices that have limited computational resources when compared to server-grade hardware that those deep-learning models were initially designed to run on. However, in the early days, when most of the traditional computer vision-based ALPR

solutions were designed, the computational resources of a PC were nearly similar to raspberry pi in modern days. Therefore, although the traditional computer vision solutions were not necessarily designed to run on edge devices, their computational resources were considerably less compared to modern-day deep learning solutions.

Therefore, a major bottleneck with the existing state-of-the-art deep learning models like RPNet and TE2E is that they were tested on powerful machines with extremely powerful GPUs. For instance, RPNet was tested on PCs with eight 3.40 GHz Intel Core i7-6700 CPU, 24 GB RAM, and one Quadro P4000 GPU. Also, TE2E was tested on an NVIDIA Titan X GPU with 12 GB memory. Thus, techniques such as RPNet cannot be deployed in edge devices.

In addition, if the inference is performed by sending the images to send to the servers, data transmission via the networks adds latency to the system. Interestingly, in this study, we have proposed Lite-LPNet, optimal learning models for license plate detection and recognition that execute on edge devices where the computational resources are limited. Thus, as stated in Table 3, while RPNet can achieve over 90% accuracy for license plate recognition, but requires server-grade hardware to run inference, the proposed method runs purely on edge-devices with accuracy competitive with the solutions designed to run on server-grade hardware.

Finally, the proposed method has been tested for both daytime and nighttime and shown empirical performance in both conditions in low resource hardware. Most of the related studies have implemented on modern hardware settings, and may not execute on edge devices with limited resources. They were also tested on powerful machines with powerful GPUs. However, the proposed method can execute on edge devices such as Raspberry pi with limited memory and power constraints, showing competing results.

6.3 | Future research directions

The design and development of the proposed approach can be extended for further experimentation and developments. One possible extension is to improve the NAS process for different hardware platforms. Although the proposed method only explored two differential NAS strategies named PC-DARTS and FBNet, this would be productive to further explore other NAS strategies. The proposed study on hardware optimized architecture search is specific to the target hardware platform, therefore, the architecture search is necessitated to repeat whenever the hardware platform changes. Given the number of hardware platforms to be n, the search time for the proposed method is $O(n)$. However, exploring a one-shot model architecture search strategy such as SMASH,[28] provides the advantage of reducing the search time to O(1) by discovering sample models that are optimized for any hardware platform.

Also, in the current study, the search space is less granular, since it is defined with a set of predefined blocks that are inspired by state-of-the-art models in both ALPR context and object detection. Therefore, in further research, the use of the NAS method to search for the cells could be a means of improving the overall architecture search process.

Second, the proposed methodology is trained using only a set of a horizontally oriented data set. However, it can be easily extended to a data set with different orientations (rotation), lighting conditions, following a similar method we used in synthetic data generation. Then the learning model can train with different variations of the images. Moreover, it would also be

compelling to evaluate the system on some other large and diverse LP data sets such as OpenALPR,[61] UFPR-ALPR,[12] and PKU data set[62] to validate the generalization of the proposed method in various complex environments.

Based on the application environment, the proposed model can also be extended to features such as identifying illegal license plates by comparing with an external data source. Moreover, in the domain of surveillance, there have been records on violating traffic laws and evading tickets with means of illegal license plates that can foil traffic cameras and modern ALPR systems. They have installed plastic covers, IR blocking stickers, and nano reflective vinyl stickers to prevent the camera from getting a clear picture of the license plate. Though it is beyond the scope of the current research, exploring further research of this is another promising direction.

7 | CONCLUSION

This study has designed and developed an ALPR system that executes on edge devices and capable of operating at night without any additional illumination. This study intended to build a system to deploy in remote uncontrolled areas without any direct access to the internet or power grid. We proposed Lite-LPNet, a family of optimal learning models for license plate detection and recognition processes that execute on edge devices and discovered by differential architecture search strategies named FBNet and PC-DARTS. The study provided a novel contribution to the ALPR context because, to the best of our knowledge, there have been no previous efforts for license plate recognition purely on edge devices with an accuracy competitive with the solutions designed to execute on server-grade hardware. However, the generalizability of these results is subject to certain limitations and the proposed methodology can be extended for future promising research directions.

CONFLICT OF INTERESTS

The authors declare that there are no conflict of interests.

ORCID

Jithmi Shashirangana https://orcid.org/0000-0001-8114-5700
Heshan Padmasiri https://orcid.org/0000-0001-9079-1747
Dulani Meedeniya https://orcid.org/0000-0002-4520-3819
Charith Perera https://orcid.org/0000-0002-0190-3346
Soumya R. Nayak https://orcid.org/0000-0002-4155-884X
Janmenjoy Nayak https://orcid.org/0000-0002-9746-6557
Shanmuganthan Vimal https://orcid.org/0000-0002-1467-1206
Seifidine Kadry https://orcid.org/0000-0002-1939-4842

REFERENCES

1. Shashirangana J, Padmasiri H, Meedeniya D, Perera C. Automated license plate recognition: a survey on methods and techniques. *IEEE Access*. 2021;9:11203-11225. https://doi.org/10.1109/ACCESS.2020.3047929
2. Caner H, Gecim HS, Alkar AZ. Efficient embedded neural-network-based license plate recognition system. *IEEE Trans Vehicular Technol*. 2008;57(5):2675-2683. https://doi.org/10.1109/TVT.2008.915524

3. Arth C, Limberger F, Bischof H. Real-time license plate recognition on an embedded DSP-platform. In: *2007 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE; 2007:1-8. https://doi.org/10.1109/CVPR.2007.383412
4. Lee S, Son K, Kim H, Park J. Car plate recognition based on CNN using embedded system with GPU. In: *2017 10th International Conference on Human System Interactions (HSI)*. IEEE; 2017:239-241. https://doi.org/10.1109/HSI.2017.8005037
5. Luo L, Sun H, Zhou W, Luo L. An ecient method of license plate location. In: *1st International Conference on Information Science and Engineering*. IEEE; 2009:770-773. https://doi.org/10.1109/ICISE.2009.250
6. Rasheed S, Naeem A, Ishaq O. Automated number plate recognition using hough lines and template matching. In: *World Congress on Engineering and Computer Science*, San Francisco, CA; 2012:24-26.
7. Nordin MJ, Ashtari A, Fathy M. An Iranian license plate recognition system based on color features. *IEEE Trans Intell Transport Syst*. 2014;15(4):1690-1705. https://doi.org/10.1109/TITS.2014.2304515
8. Shyang-Lih C, Li-Shien C, Yun-Chung C, Sei-Wan C. Automatic license plate recognition. *IEEE Trans Intell Transport Syst*. 2004;5(1):42-53. https://doi.org/10.1109/TITS.2004.825086
9. Yu S, Li B, Zhang Q, Liu C, Meng MQH. A novel license plate location method based on wavelet transform and EMD analysis. *Pattern Recognition*. 2015;48(1):114-125. https://doi.org/10.1016/j.patcog.2014.07.027
10. Giannoukos I, Anagnostopoulos CN, Loumos V, Kayafas E. Operator context scanning to support high segmentation rates for real time license plate recognition. *Pattern Recogn*. 2010;43(11):3866-3878. https://doi.org/10.1016/j.patcog.2010.06.008
11. Zhou W, Li H, Lu Y, Tian Q. Principal visual word discovery for automatic license plate detection. *IEEE Trans Image Process*. 2012;21(9):4269-4279. https://doi.org/10.1109/TIP.2012.2199506
12. Laroca R, Severo E, Zanlorensi L, et al. A robust real-time automatic license plate recognition based on the YOLO Detector. In: *International Joint Conference on Neural Networks (IJCNN)*; 2018:1-10. https://doi.org/10.1109/IJCNN.2018.8489629
13. Hsu GS, Ambikapathi AM, Chung SL, Su CP. Robust license plate detection in the wild. In: *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*; 2017:1-6. https://doi.org/10.1109/AVSS.2017.8078493
14. Xie L, Ahmad T, Jin L, Liu Y, Zhang S. A new CNN-based method for multi-directional car license plate detection. *IEEE Trans Intell Transport Syst*. 2018;19(2):507-517. https://doi.org/10.1109/TITS.2017.2784093
15. Redmon J, Farhadi A. YOLOv3: an incremental improvement. Tech Report. 2018 arxiv:1804.02767.
16. Xu Z, Yang W, Meng A, et al. Towards end-to-end license plate detection and recognition: a large dataset and baseline. In: *European Conference on Computer Vision (ECCV)*; 2018:261-277. https://doi.org/10.1007/978-3-030-01261-8_16
17. Li H, Wang P, Shen C. Toward end-to-end car license plate detection and recognition with deep neural networks. *IEEE Trans Intell Transport Syst*. 2019;20(3):1126-1136. https://doi.org/10.1109/TITS.2018.2847291
18. Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. 2015:1-14. arxiv:1409.1556v6
19. Alborzi Y, Mehraban TS, Khoramdel J, Ardekany AN. Robust real time lightweight automatic license plate recognition system for Iranian license plates. In: *2019 7th International Conference on Robotics and Mechatronics (ICRoM)*; 2019:352-356. https://doi.org/10.1109/ICRoM48714.2019.9071863
20. Zherzdev S, Gruzdev A. LPRNet: license plate recognition via deep neural networks. *CoRR*; 1-16. arxiv:1806.10447.
21. Izidio DMF, Ferreira APA, Medeiros HR, Silva Barros dEN. An embedded automatic license plate recognition system using deep learning. *Des Autom Embed Syst*. 2020;24(1):23-43. https://doi.org/10.1007/s10617-019-09230-5
22. Elsken T, Metzen JH, Hutter F. Neural architecture search: a survey. *J Mach Learn Res*. 2019;20:63-77. https://doi.org/10.1007/978-3-030-05318-5_3
23. Liu H, Simonyan K, Yang Y. DARTS: differentiable architecture search. *In: International Conference on Learning Representations (ICLR 2019)*. 2018:1-13. arxiv:1806.09055.
24. Xu Y, Xie L, Zhang X, et al. PC-DARTS: partial channel connections for memory-efficient architecture search. 2020:1-13. arXiv:1907.05737.

25. Wu B, Dai X, Zhang P, et al. FBNet: hardware-aware efficient convnet design via differentiable neural architecture search Computer Vision Foundation. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Long Beach, CA: IEEE; 2019:10734-10742. https://doi.org/10.1109/CVPR.2019.01099
26. Zoph B. Neural architecture search with reinforcement learning. In: *5th International Conference on Learning Representations (ICLR)*; 2019:1-16.
27. Liu C, Zoph B, Neumann M, et al. Progressive neural architecture search. In: *European Conference on Computer Vision (ECCV)*; 2018:19-34. https://doi.org/10.1007/978-3-030-01246-5_2
28. Brock A, Lim T, Ritchie JM, Weston N. SMASH: one-shot model architecture search through hypernetworks. In: *6th International Conference on Learning Representations (ICLR)*. Vancouver, BC, Canada: OpenReview.net; 2018:1-22.
29. Baker B, Gupta O, Raskar R, Naik N. Accelerating Neural Architecture Search using Performance Prediction. In: *6th International Conference on Learning Representations (ICLR)*. Vancouver, BC, Canada: OpenReview.net; 2018:1-19.
30. Saxena S, Verbeek J. Convolutional neural fabrics. In: Lee DD, Sugiyama M, Luxburgv U, Guyon I, Garnett R, eds. *Advances in Neural Information Processing Systems, 29th Annual Conference on Neural Information Processing Systems*. Barcelona, Spain; 2016:4053-4061.
31. Ahmed K, Torresani L. Connectivity learning in multi-branch networks. 2017:1-17. arXiv:1709.09582.
32. Ghiasi G, Lin TY, Pang R, Le QV. NAS-FPN: learning scalable feature pyramid architecture for object detection. 2019:1-10. arXiv:1904.07392.
33. Xu H, Yao L, Li Z, Liang X, Zhang W. Auto-FPN: automatic network architecture adaptation for object detection beyond classification. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*; 2019:6648-6657. https://doi.org/10.1109/ICCV.2019.00675
34. Tan M, Pang R, Le QV. EfficientDet: scalable and efficient object detection. In: *Conference on Computer Vision and Pattern Recognition (CVPR)* IEEE; 2020:10778-10787. https://doi.org/10.1109/CVPR42600.2020.01079
35. Tan M, Le QV. EfficientNet: rethinking model scaling for convolutional neural networks. In: *36th International Conference on Machine Learning (ICML)*; 2019:1-11.
36. Du X, Lin TY, Jin P, et al. SpineNet: learning scale-permuted backbone for recognition and localization. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE; 2020:11589-11598. https://doi.org/10.1109/CVPR42600.2020.01161
37. Chen Y, Yang T, Zhang X, Meng G, Xiao X, Sun J. DetNAS: backbone search for object detection. In: Wallach HM, Larochelle H, Beygelzimer A, Buc dF, Fox EB, Garnett R, eds. *Conference on Neural Information Processing Systems (NeurIPS)*; 2019:6638-6648.
38. Hong S, Kim D, Choi M. Memory-efficient models for scene text recognition via neural architecture search. In: *IEEE Winter Applications of Computer VisionWorkshops (WACV)*. Snowmass Village, CO: IEEE; 2020:183-191. https://doi.org/10.1109/WACVW50321.2020.9096928
39. Zhao Z, Jiang M, Guo S, Wang Z, Chao F, Tan KC. Improving deep learning based optical character recognition via neural architecture search. In: *IEEE Winter Applications of Computer VisionWorkshops (WACV)*. Snowmass Village, CO: IEEE; 2020:1-7. https://doi.org/10.1109/CEC48606.2020.9185798
40. Zoph B, Vasudevan V, Shlens J, Le QV. Learning transferable architectures for scalable image recognition. In: *IEEE conference on computer vision and pattern recognition*; 2018:8697-8710. https://doi.org/10.1109/CVPR.2018.00907
41. Howard AG, Zhu M, Chen B, et al. Mobilenets: efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861. 2017.
42. Sandler M, Howard A, Zhu M, Zhmoginov A, Chen LC. Mobilenetv2: inverted residuals and linear bottlenecks. In: *IEEE conference on computer vision and pattern recognition*; 2018:4510-4520. https://doi.org/10.1109/CVPR.2018.00474
43. Tan M, Chen B, Pang R, et al. MnasNet: platform-aware neural architecture search for mobile. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Long Beach, CA: IEEE; 2019:2820-2828. https://doi.org/10.1109/CVPR.2019.00293
44. Zhang LL, Yang Y, Jiang Y, Zhu W, Liu Y. Fast hardware-aware neural architecture search. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE; 2020:2959-2967. https://doi.org/10.1109/CVPRW50498.2020.00354

45. Sandler M, Howard A, Zhu M, Zhmoginov A, Chen L. MobileNetV2: inverted residuals and linear bottlenecks. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*; 2018:4510-4520. https://doi.org/10.1109/CVPR.2018.00474
46. Szegedy C, Ioffe S, Vanhoucke V, Alemi AA. Inception-v4, inception-ResNet and the impact of residual connections on learning. In: Singh SP, Markovitch S., eds. *31st AAAI Conference on Artificial Intelligence*. San Francisco, CA: AAAI Press; 2017:4278-4284.
47. Loshchilov I, Hutter F. SGDR: stochastic gradient descent with restarts. CoRR; 2016:1-16. arXiv:1608.03983
48. Krizhevsky A. Learning Multiple Layers of Features from Tiny Images. Technical Report TR-2009, University of Toronto, Toronto; 2009:32-33.
49. Deng J, Dong W, Socher R, Li L, Li K, Li F. ImageNet: a large-scale hierarchical image database. In: *IEEE Computer Society*; 2009:248-255. https://doi.org/10.1109/CVPR.2009.5206848
50. He K, Gkioxari G, Dollár P, Girshick RB. Mask R-CNN. *IEEE Trans Pattern Anal Mach Intell*. 2020;42(2):386-397. https://doi.org/10.1109/TPAMI.2018.2844175
51. Zhang L, Gonzalez-Garcia A, Weijer vdJ, Danelljan M, Khan FS. Synthetic data generation for end-to-end thermal infrared tracking. *IEEE Trans Image Process*. 2019;28(4):1837-1850. https://doi.org/10.1109/TIP.2018.2879249
52. Hwang S, Park J, Kim N, Choi Y, Kweon IS. Multispectral pedestrian detection: Benchmark dataset and baseline. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Boston, MA: IEEE Computer Society; 2015:1037-1045. https://doi.org/10.1109/CVPR.2015.7298706
53. Isola P, Zhu J, Zhou T, Efros AA. Image-to-image translation with conditional adversarial networks. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI: IEEE Computer Society; 2017:5967-5976. https://doi.org/10.1109/CVPR.2017.632
54. Padmasiri H, Madurawe R, Abeysinghe C, Meedeniya D. Automated vehicle parking occupancy detection in real-time. In: *Moratuwa Engineering Research Conference (MERCon)*; 2020:644-649. https://doi.org/10.1109/MERCon50084.2020.9185199
55. Wong T, Yeh P. Reliable accuracy estimates from k-fold cross validation. *IEEE Trans Knowl Data Eng*. 2020;32(8):1586-1594. https://doi.org/10.1109/TKDE.2019.2912815
56. Jamtsho Y, Riyamongkol P, Waranusast R. Real-time Bhutanese license plate localization using YOLO. *ICT Express*. 2020;6(2):121-124. https://doi.org/10.1016/j.icte.2019.11.001
57. Laroca R, Zanlorensi LA, Gonçalves GR, Todt E, Schwartz WR, Menotti D. An efficient and layout-independent automatic license plate recognition system based on the YOLO detector; 2020:1-18. arXiv:1909.01754
58. Sarfraz M, Ahmed MJ, Ghazi SA. Saudi Arabian license plate recognition system. In: *International Conference on Geometric Modeling and Graphics*. IEEE; 2003:36-41. https://doi.org/10.1109/GMAG.2003.1219663
59. Llorens D, Marzal A, Palazón V, Vilar JM. Car license plates extraction and recognition based on connected components analysis and HMM decoding. In: *Iberian conference on pattern recognition and image analysis*. Springer; 2005:571-578. https://doi.org/10.1007/11492429_69
60. Selmi Z, BenHalima M, Alimi AM. Deep learning system for automatic license plate detection and recognition. In: *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*. 2017;1:1132-1138. https://doi.org/10.1109/ICDAR.2017.187
61. Inc. O. OpenALPR-EU dataset. https://github.com/openalpr/benchmarks/tree/master/endtoend/eu; 2016.
62. Yuan Y, Zou W, Zhao Y, Wang X, Hu X, Komodakis N. A robust and efficient approach to license plate detection. *IEEE Trans Image Process*. 2017;26(3):1102-1114. https://doi.org/10.1109/TIP.2016.2631901

How to cite this article: Shashirangana J, Padmasiri H, Meedeniya D, et al. License plate recognition using neural architecture search for edge devices. *Int J Intell Syst*. 2021; 1-38. https://doi.org/10.1002/int.22471