Received: 30 August 2022 | Revised: 14 February 2023 | Accepted: 15 May 2023

DOI: 10.1111/exsy.13358

ORIGINAL ARTICLE

Expert Systems WILEY

Improving the accuracy and speed of fast template-matching algorithms by neural architecture search

Seyed Mahdi Shariatzadeh | Mahmood Fathy | Reza Berangi

Department of Computer Engineering, IRAN University of Science and Technology (IUST), Tehran, Iran

Correspondence
Mahmood Fathy, Department of Computer Engineering, IRAN University of Science and Technology (IUST), Tehran, Iran.
Email: mahfathy@iust.ac.ir

Abstract

Neural architecture search can be used to find convolutional neural architectures that are precise and robust while enjoying enough speed for industrial image processing applications. In this paper, our goal is to achieve optimal convolutional neural networks (CNNs) for multiple-templates matching for applications such as licence plates detection (LPD). We perform an iterative local neural architecture search for the models with minimum validation error as well as low computational cost from our search space of about 32 billion models. We describe the findings of the experience and discuss the specifications of the final optimal architectures. About 20-times error reduction and 6-times computational complexity reduction is achieved over our engineered neural architecture after about 500 neural architecture evaluation (in about 10 h). The typical speed of our final model is comparable to classic template matching algorithms while performing more robust and multiple-template matching with different scales.

KEYWORDS

convolutional neural networks, neural architecture search, multi-objective optimization, object detection, pattern matching, licence plate detection

1 | INTRODUCTION

Despite having relatively good accuracy and robust performance, one of the most critical problems of traditional convolutional neural networks (CNNs) in embedded applications is their high computational cost. The two-dimensional convolution operation for multiple channels is a relatively costly operator which makes it hardly practical for industrial and embedded applications without hardware acceleration. Many research works and techniques are focused to suppress this issue, notably structural and weight pruning and quantization (Liang et al., 2021), separable or the strided convolutions (Chollet, 2017a), global pooling layers (Springenberg et al., 2014), and other improvements that tend to reduce the computations in the CNNs while keeping their benefits. In this paper, we propose a method based on neural architecture search (NAS) to find a model with a minimum validation error and low computation cost for an embedded application, such as licence plate recognition (LPR).

Usual LPR systems consist of the two stages of LPD and optical character recognition (OCR). For the OCR stage, there are methods such as segmentation-free deep LPR (Bulan et al., 2017; Weihong & Jiaoyang, 2020) which typically use recurrent neural networks (RNNs) trained with the connectionist temporal classification (CTC) loss. However, the earlier stage of licence plate detection is still challenging, especially when considering both speed and precision. Although there are some end-to-end LPR methods that perform the detection and recognition (OCR) stages together (Jaderberg et al., 2015), they are not good for industrial/embedded usage, because: (1) typical end-to-end methods suffer from high computational costs; (2) these methods are not adjustable and tunable for different installation setups; and (3) they are hardly explainable and perform as a black-box, which is not good for industrial engineering and development. Therefore, we prefer to develop separated LPD and OCR stages.

In our application region, there are two different templates for legal licence plates (National and Free-Zone); the plate sizes in the images vary due to the perspective, and some possible small rotation and tilt, blurriness, dirty plates, cracks and deformations. Also, there may be several licence plates in one input image.

In this situation, it can act as an early stage of the LPR and optical speed estimation, for example, in an intelligent transportation system (ITS). The final algorithm should be good-enough for replacing famous object detection CNNs such as YOLO as well as heuristic methods and fast template matching algorithms such as Viola-and-Jones cascade classifiers (Viola & Jones, 2001).

Our goal is to develop a CNN for licence-plate detection that is needed to be fast and run in real time while having high precision and being robust to input degradations. As there is a lot of possible structures and hyper-parameters, it is possible to do a search in the space of all possible neural architectures, which is the concept of NAS (Zoph et al., 2018). The NAS is an optimization process through the network architecture and hyperparameters for a single-objective (i.e., accuracy) or multiple-objectives (such as any combinations of accuracy, model size, inference latency, etc.). The optimization methods for NAS can be categorized in gradient-based methods (GR), evolutionary algorithms (EA), reinforcement learning (RL), or heuristic search (HS) algorithms (including greedy algorithms and random search). Popular NAS methods include differentiable architecture search (DARTS; Liu, Simonyan, & Yang, 2018), Efficient NAS via Parameters Sharing (Pham et al., 2018), Progressive NAS (Liu, Zoph, et al., 2018), and Platform-Aware Neural Architecture Search for Mobile (MnasNet; Tan et al., 2019) incorporates both accuracy and latency on mobile hardware. The survey in Elsken et al. (2019) provides a good review on the most important NAS methods.

We use an application-specific NAS to optimize our baseline CNN for matching multiple templates in the images. We define a network architecture template having 14 integer parameters which specially considers arbitrary-shaped strided convolutions and customized image downsizing. Our NAS experience is based on the *Iterated Local Search*. This method lies in the category of greedy algorithms (heuristic search) with a mechanism to escape local minima and continue the search for further answers. Also, it can be considered as a very special case of evolutionary algorithm with a population size of 1 and no cross-over operation. Once detected, the final model is easily transferable to multiple licence plates detection on any arbitrary-size images.

The major contributions of our paper are as follows: (1) Performing multi-objective NAS for multiple template matching in images, considering low error rate as well as low computational cost for embedded applications. (2) Providing further analysis on the Pareto-optimum resulting architectures. (3) A simple-yet-efficient architecture search space for object detection, image classification and template matching.

The rest of this paper is organized as follows: In Section 2, we define our notation and problem formulation for NAS. In the next section, the hyper-architecture for object detection is presented. Then, the results of our NAS experience are explained in Section 4 along with comparisons to other methods. We conclude our paper with analysis and an abstract list of key findings on the best found architectures.

2 | OUR NEURAL ARCHITECTURE SEARCH FORMULATION AND METHOD

For the NAS problem definition, we follow the notations of (Jin et al., 2019): For the *training and validation datasets* D_{train} and D_{val}, we are searching for *the best architecture* f^* in *the neural architecture search space* F which minimizes the *cost function* $Cost(.,.)$ for the best trained *network parameters* θ, learned by the *learning algorithm* $L(.,.)$ (Jin et al., 2019):

$$f^* = \underset{f \in F}{\arg\min}\ Cost(f(\theta^*), D_{val}), \tag{1}$$

$$\theta^* = \underset{\theta}{\arg\min}\ L(f(\theta), D_{train}). \tag{2}$$

In this work, we further continue the above notation as follows: The network architecture f is defined by *describing parameters vector* p and *the network architecture generator* G:

$$f = G(p). \tag{3}$$

Any changes in a single variable p_i of parameter p to p' make a structure-adjustment in architecture $f = G(p)$ to f'.

As we are searching for a lightweight model with a low error rate, we have two primary objectives. At first, we scalarize the cost function of our multi-objective optimization problem in Equation (1) as a linear combination of the validation error of the network and its computational cost:

$$Cost(f,.) = Cost_{Error}(f,.) + \alpha.Cost_{Complexity}(f,.), \tag{4}$$

$$Cost(f, D_{val}) = Err_{D_{val}}(f) + \alpha.\frac{CC_I(f)}{w(I).h(I)}, \tag{5}$$

in which, $CC_I(f)$ is the computational cost of the network f for a typical input I (e.g., in [*FLOP/frame*]) that, for ease of analyses, is scaled by the Input size (i.e., in [*FLOP/px*]) and α is the factor for the Error-versus-Computational-cost trade-off. The α factor should be set according to the problem and the dataset.

The *Local Search* Optimizations are heuristic methods to find candidate answers by altering the input parameters and evaluating the neighbour possible solutions. They tend to a local-optimum solution. The Iterative Local Search methods are their generalizations that try to avoid trapping in local-optima by severely perturbing the parameters when no improvements seen in the neighbourhood of the latest candidate. We will use the iterative local search in the space of CNN architectures to find desirable networks.

3 | PROPOSED CNN META-ARCHITECTURE

We need a parametric CNN meta-architecture to generate a good variety of CNNs for our application. First, we consider a primary model: a CNN should consist of three consequence stages: (1) preprocessing; (2) convolutional stage; and (3) the output-generation stage. The convolutional stages are a sequence of single *Convolutional Building-Block*s. Each Convolutional Building-Block consists of one or two parallel separable convolution(s) that are channel-wise concatenated, followed by an Activation Layer. We also use Batch-Normalization and Layer-Normalization for better convergence only on the first convolutional stage. Figure 1 illustrates our neural architecture template. Putting all together, there is a total of 14 hyperparameters in our hyper-architecture.

We prefer the separable convolution operators to the normal ones for their lower computational cost. Also, the convolution operations can have strides, possibly different in height and width dimensions. Finally, the output-generation stage differs in training and usage times; after two convolutions to produce a single-channel matrix, for training, it ends with a parameterless *Global Max Pooling* layer; after training, we replace it with a *Non-Maximum-Suppression* layer to detect possibly multiple plates in the input image. Table 1 shows the structure of our architecture template.

Once the best architecture has been found, there will be a simple process to handle full-frame inputs and multiple licence plates detection: Note that our search space is restricted to a class of networks that are ready to scale on the input only by replacing the final layer: We feed the network with full-frame inputs and replace the final global max pooling layer with non-maximum suppression. The crop scales can be set as the input size of the architectures in the search space. We do not require finding a tight boundary as the OCR stage can tolerate misalignments in the spatial cropping.

It worth noting that avoiding join-detection and recognition results in high computational efficiency, while the mid-level features may further be used for recognition, too.

4 | EXPERIMENTAL RESULTS

4.1 | The experimental setup

We have used a dataset consisting images from both templates of the plates as well as non-plate images. The non-plate images are made by random crops from 10 traffic images (without plates), further having random transformations such as rotation, aspect ratio variation and blur. Each sample is

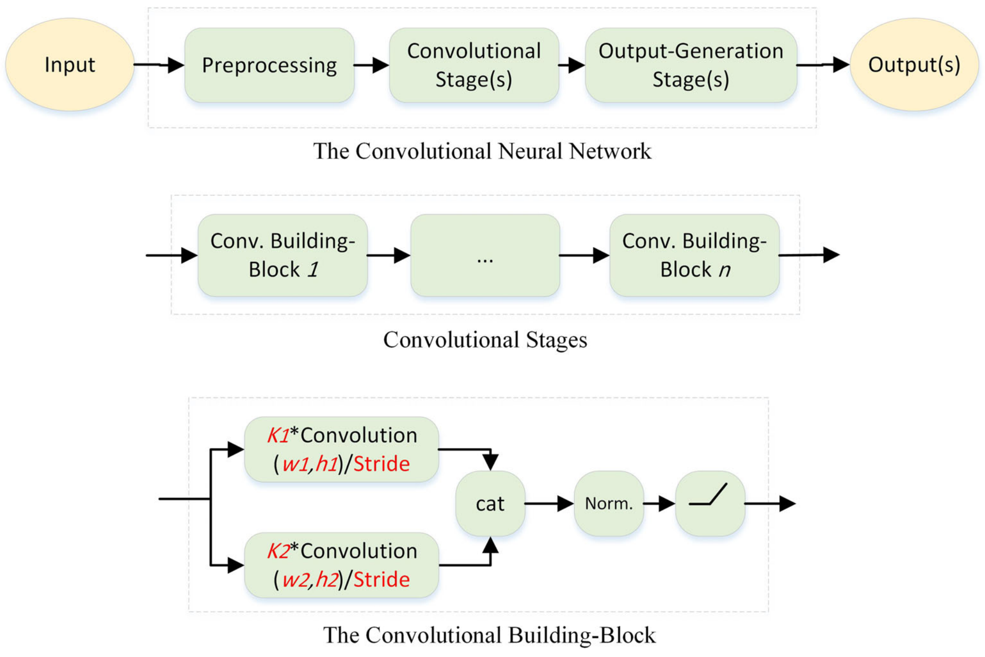
Input
Preprocessing
Convolutional Stage(s)
Output-Generation Stage(s)
Output(s)
The Convolutional Neural Network
Conv. Building-Block 1
...
Conv. Building-Block n
Convolutional Stages
K1*Convolution (w1,h1)/Stride
K2*Convolution (w2,h2)/Stride
cat
Norm.
The Convolutional Building-Block

FIGURE 1 The proposed template of a general convolutional neural network, as used in our neural architecture search algorithm. Each component may have hyper-parameters as denoted in red. A total of 14 hyper-parameters exists in our neural architecture search hyper-architecture.

TABLE 1 The structure of our hyper-architecture.

Layer#	Stage	Layer type	Parameters
1	Preprocessing	Downsampling	(h_D, w_D)
2.1	Conv.	Separable Convolution A	$k_A, (h_A, w_A)/st_1$
2.2	Conv.	Separable Convolution B	$k_B, (h_B, w_B)/st_1$
2.3	Conv.	Batch Normalization	
2.4	Conv.	Layer Normalization	
2.5	Conv.	Activation Layer	
$i : 2..k_s$			
$i.1$	Conv.	Separable Convolution A	$k_A, (h_A, w_A)/st_a$
$i.2$	Conv.	Separable Convolution B	$k_B, (h_B, w_B)/st_a$
$i.3$	Conv.	Activation Layer	
$k_s + 2$	Output Gen.	Separable Convolution	$k_O, (h_O, w_O)/1$
$k_s + 3$	Output Gen.	Depthwise Convolution	1, (1,1)/1
$k_s + 4$	Output Gen.	Global Max-Pooling/NonMax. Suppression	

grayscale (to be capable of being used for both Infrared and visible-colour images) with a resolution of 190×60 pixels. In the samples having plates, the width of the plates varied from 80 px (the least width for an image of a dirty plate to be readable) up to 190 px (the width of the biggest plate image in a usual ITS camera setup). The datasets are consisted of $n(D_{train}) = 30k$ and $n(D_{val}) = 1k$ images, about 20% of which are objects and the rests are background images. Figure 2 shows samples from the dataset and the background images used. The source code for generating non-plate images as well as the background images and the validation set are publicly available at www.github.com/shariatzadeh/IranianLPD-NAS.

At each epoch, data augmentation is done by slight geometrical transformations such as rotation, scaling, shift, and changing the aspect ratio. For 30%-random samples an application-specific data augmentation are performed by adding random amounts of blur, Gaussian additive noise, brightness/contrast change, and JPEG compression noise with a random severity. Note that the architectures in our search space are intrinsically translation invariant.

4.2 | The results and specifications of the NAS experiment

The overall NAS experiment is conducted in about 10 h. During this procedure 522 models were searched, with 385 distinct models each trained once.

The convergence trend is shown in Figure 3. Without the local-minima exit mechanism, the algorithm would be stopped at about 30th iteration.

In Figure 3, it can be seen that the two terms of Equation (4) have relatively near values for the intermediate best results. We can use this intuition to set the α term automatically: the term α should balance the Error/Complexity costs on the observed distribution. Therefore, considering the evaluation of the initialization model f^0, we propose:

$$\alpha = \frac{Cost_{Error}\left(f^0,.\right)}{Cost_{Complexity}\left(f^0,.\right)}, \tag{6}$$

that means we pay benefits to optimizing both $Cost_{Error}(.,.)$ and $Cost_{Complexity}(.,.)$ for the initialization model equally. Note that this equation makes the whole process invariant to scaling factors of the measurements, so that there is no difference in, for example, measuring $Cost_{Complexity}\left(f^0,.\right)$ by $[FLOP]$, $[MFLOP]$, $[FLOP/px]$ and so on. On the other hand, we can see in Figure 4b that the α term is not sensitive: a more than %20 alternation is needed to change the returning lowest cost model.

4.3 | The resulting architectures for the DCNNs

As described above, a multi-objective optimization problem usually leads to a set of Pareto-optimal answers, each of which is a dominant answer in the trade-off of different objectives (in our experiment, the accuracy and the computational cost of the model). As shown in Figure 4, we have 10 Pareto-optimal neural architectures (illustrated on the Pareto frontier) as the result of the NAS process.

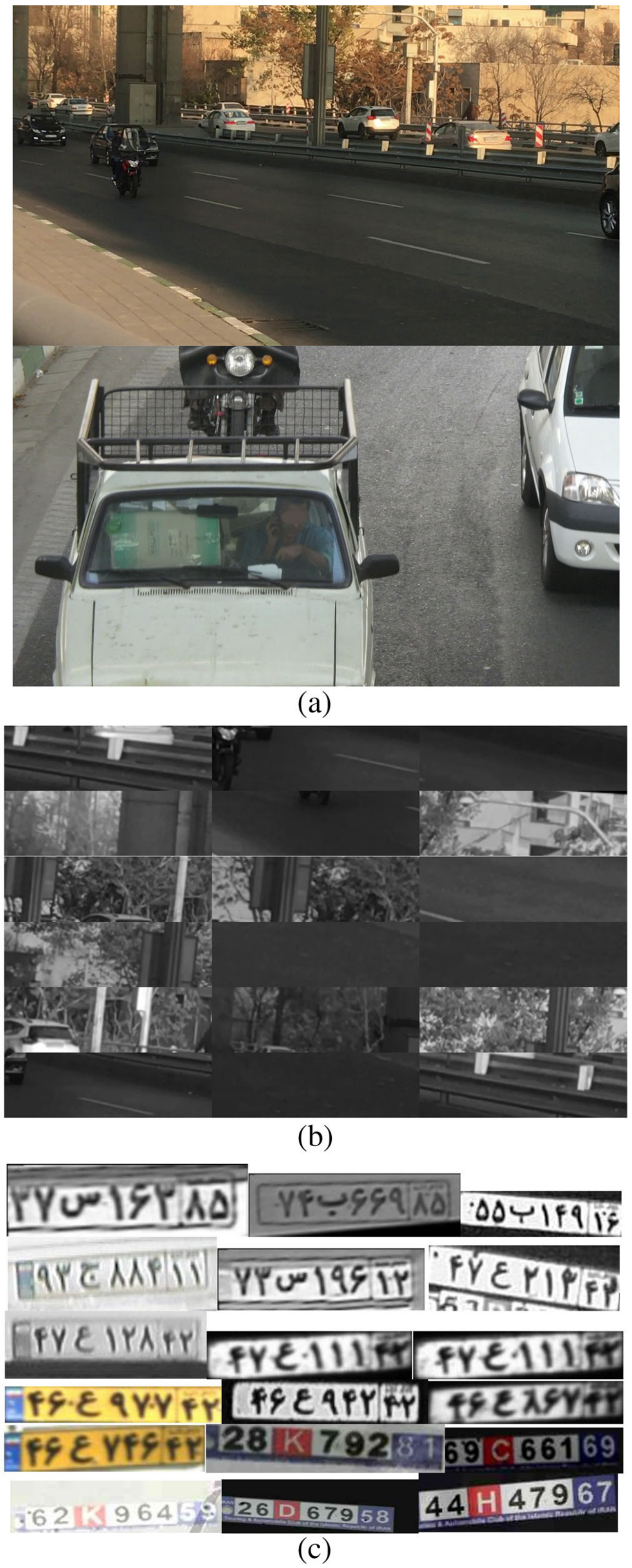

FIGURE 2 The dataset used in our experiments; (a) sample background images from the ITS domain (see text for the description); (b) samples of non-plate images extracted from background images; (c) samples of plate images.

The hyperparameters of the architecture with the least scalarized cost (according to Equation (4) and Tables 1 and 2) are:

$$(h_D, w_D) = (2, 2),$$

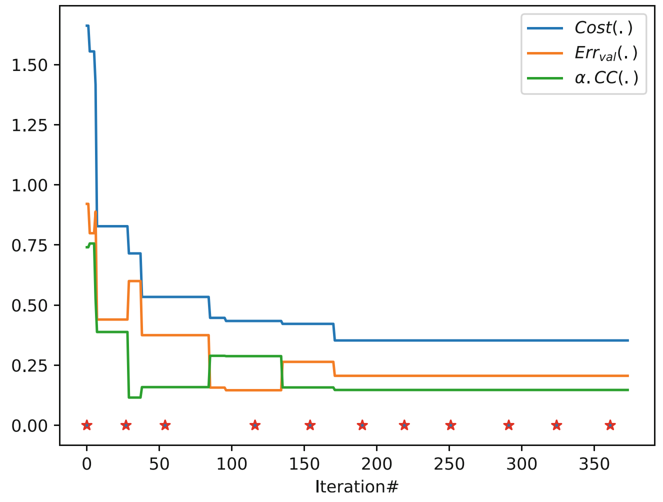

FIGURE 3 The neural architecture optimization trend through time. The red stars denote the severe perturbations of the optimization algorithm to jumping out of local minima.

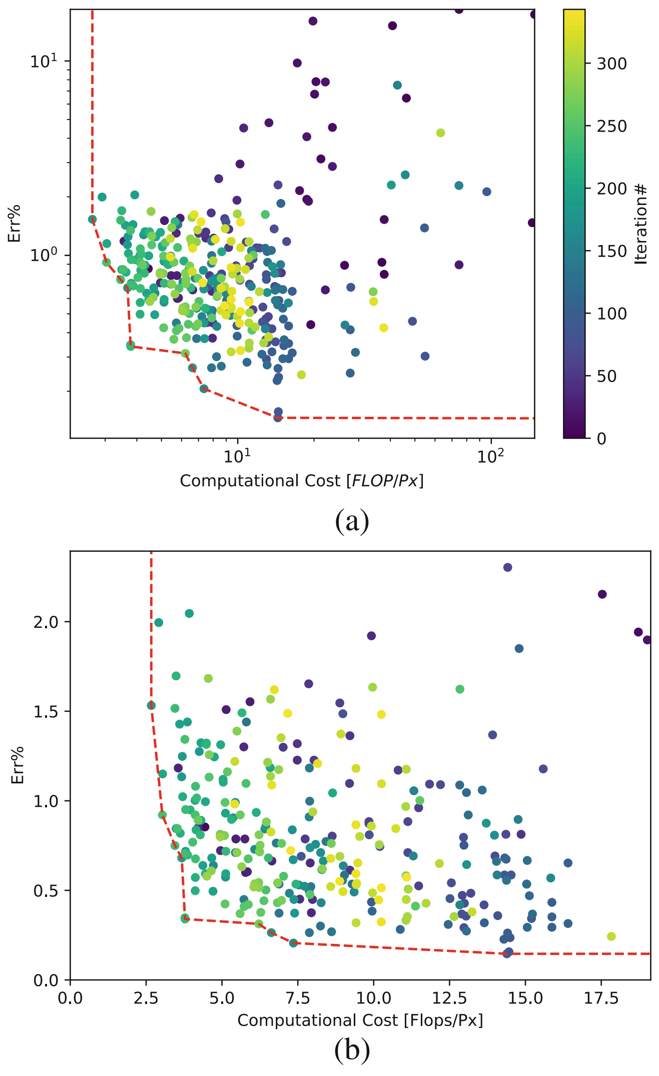

FIGURE 4 Illustration of the Error-versus-Complexity attribute for each of the trained networks in our NAS experiment; (a) the whole plot in log–log scales; (b) the partial near-zero part. The *Pareto Frontier* is denoted by the red dashed curve. The colour of each point from blue to green and yellow illustrates the NAS iteration number.

TABLE 2 The description of the items in our network hyperparameter vector *p*.

Param#	Stage	Parameter	Description
1, 2	Preprocess.	(h_D, w_D)	Horizontal and vertical downsampling ratios
3	Conv.	k_s	Number of the convolutional stages
4	Conv.	st_1	Amount of the stride for the first stage
5	Conv.	st_a	Amount of the stride for the rest of stages
6, 7	Conv.	k_A, k_B	Number of convolutional filters for A and B
8, 9	Conv.	(h_A, w_A)	Filter size of the convolution A
10, 11	Conv.	(h_B, w_B)	Filter size of the convolution B
12	Output Gen.	k_O	Number of convolutional filters for O
13, 14	Output Gen.	h_O, w_O	Filter size of the convolution O

$$k_s = 4,$$

$$st_1 = 3,$$

$$st_a = 2,$$

$$k_A, k_B = 3,3$$

$$(h_A, w_A) = (5,3),$$

$$(h_B, w_B) = (2,4),$$

$$k_O = 7,$$

$$(h_O, w_O) = (2,5).$$

This architecture (indicated in Table 4 as *Ours-2*) has four layers of concurrent convolution, followed by a separable convolution, depthwise convolution, and the final global max-pooling layer.

The above architecture reaches $err = 0.206\%$ with the computational cost of 7.356 $[Ops/Px]$. It enjoys only 387.0 k parameters which is relatively fast, lightweight and accurate. Note that other optimal solutions may have less computational cost with more detection error (or vice-versa).

I worth noting that the aforementioned architecture has several interesting specifications in comparison to the most well-known architectures such as GoogleNet, MobileNet and DenseNet: In different layers, it performs convolutions with only three and four filters, except the final convolution with seven filters, while the usual number of convolution channels is 8, 32 or even more. The filter sizes are also interesting: 2×4, 5×3 and 2×5; according to the intrinsic properties of a licence plate that has a width about four times its height, the filter sizes tend to landscape. Especially, the final convolution slightly resembles the shape of a licence plate and is large-enough to match a small template of a licence plate. While having very low number of filters, there are $4 + 1$ convolution stages in the architecture which is a relatively large number for the small size of the input images. It also performs a 2×2 down-sampling as a preprocessing stage that makes it faster while does not eliminate much details from the input.

We have provided the specifications of all the Pareto-Optimal architectures in Table 3. This table shows the properties of each architectural parameter at the initial value, the values of the parameter among the Pareto Set, and the $Mode(.)$ of the parameter in the Pareto Set. It can be seen that most of our initial architectural parameters are altered by the NAS procedure, which means it is not trapped in a near local minimum.

4.4 | Comparisons

We provide a comparison between the mid-point of our Pareto-Set as well as our baseline model to other algorithms. The well-known Viola and Jones template detection algorithm that is widely used in Industrial LPR for single and multiple plate size, as well as YOLOv3 (Redmon &

TABLE 3 Results of our method for LPD: for each parameter p_i, the initial value p_i^0 from the baseline network, the values and the number of occurrences (written as superscript) in the Pareto Set—$p^{PS} = \{p^k : G(p^k) \in PS\}$, and the *Mode* on the Pareto Set $Mode(p_i^{PS})$.

i	p_i	p_i^0	$\{p_i^{PS})\}$	$Mode(p_i^{PS})$
1, 2	(h_D, w_D)	(2, 2)	$(1^{(1)}\ 2^{(4)}\ 3^{(5)},\ 1^{(1)}\ 2^{(9)})$	(3, 2)
3	k_s	3	$3^{(4)}4^{(6)}$	4
4	st_1	2	$2^{(2)}3^{(8)}$	3
5	st_a	1	$1^{(2)}2^{(5)}3^{(3)}$	2
6, 7	k_A, k_B	2, 1	$3^{(10)}, 3^{(7)}4^{(2)}5^{(1)}$	3, 3
8, 9	(h_A, w_A)	(3, 4)	$\left(3^{(5)}4^{(4)}5^{(1)}, 4^{(6)}5^{(4)}\right)$	(3, 4)
10, 11	(h_B, w_B)	(1, 4)	$\left(0^{(1)}1^{(2)}2^{(7)}, 3^{(1)}4^{(8)}5^{(1)}\right)$	(2, 4)
12	k_O	8	$7^{(4)}8^{(6)}$	6
13, 14	(h_O, w_O)	(3, 6)	$\left(1^{(5)}2^{(4)}3^{(1)}, 5^{(8)}6^{(2)}\right)$	(1, 5)

TABLE 4 Comparison of some candidate methods for licence plate detection (the best value of each column is written in bold).

	Comput. cost	Classes#	Detection
Model	$[Ops/Px]$		Err. [%]
Viola & Jones (Viola & Jones, 2001) (single scale)	~ 6	1	~ 6
Viola & Jones (Viola & Jones, 2001) (3 scales)	~ 18	1	~ 2
Our Baseline	26	2	4.3
Ours	**3.4**	2	0.4
Ours-2	7.4	2	**0.2**
YOLOv4-tiny (Bochkovskiy et al., 2020) $\downarrow (2 \times 2)$	8318	80	~ 1
YOLOv3 (Redmon & Farhadi, 2018) $\downarrow (2 \times 2)$-tiny mobilenet	1961	80	~ 1

Note: The values shown with the approximate mark are estimations for our dataset. (Note that the operations in the Viola & Jones method are in short integer, while the rest of the methods are in floating points.)

Farhadi, 2018) and YOLOv4 (Bochkovskiy et al., 2020; with speed-up by a 2×2 down-sampling preprocessing) are taken into account. Table 4 shows the specifications of the above algorithms for our application. The computational cost is reported by $[Ops/Px]$.

4.5 | Hardware and software setup

We chose the *RMSProps* optimization algorithm with default parameters and the *Binary Cross-Entropy* loss function for training all the models. The training epochs was fixed to 3 and the batch size was 32 in all of our experiments. Training each model took about 90 s.

All the convolution operations are Depth-Wise Separable (Chollet, 2017b). The Non-linearity is provided by $LeakyRELU(.,0.2)$ activation function at the end of each convolutional stage, which avoids *dead neurons* that may be made by bad random initialization and will cause performance loss, especially on low-number of neurons. We set $\alpha = 0.02$ in our experience.

We perform our experiments on a Windows 10 system using python, the Keras library (Tensorflow backend). For the optimization, we extended the *pymoo* (Multi-objective Optimization in Python) library by implementing the Iterative Local Search algorithm. All the experiments are run on a Geforce GTX 1080Ti graphic card (having 1024 cores), an Intel Core i7 CPU and 16 GB RAM.

5 | DISCUSSION AND HIGHLIGHT

We continue the analysis of our results as a multi-objective optimization through its *Pareto Set*. For a multi-objective optimization problem, the Pareto Set is the set of all answers that, for each one, there is no other better answer in the objectives. Figure 4 illustrates the properties of each found network during the NAS procedure. As can be seen, each trained network has a specification in the Error/Computational-Cost trade-off; but, according to the *Pareto Frontier* of our optimization, only a few networks are Pareto-Optimal and there is no use in choosing the other networks. Precisely, 10 of 385 evaluated networks are Pareto-Optimal, that provide performances from (1.5% Error, $2.6 FLOP/Px$) to (0.14% Error, $14.4 FLOP/Px$). The first Pareto-Optimal architecture was found at iteration 96 and the last one was at iteration 243. More than 100 last steps did not find any optimal architectures.

5.1 | Architectural analysis of the networks in the Pareto Set

Our enumerable findings from the optimal architectures are:

1. For our problem, almost all the optimum architectures utilize relatively large strides of 2 and 3.
2. Few filters (6–8) for each stage are preferred in all the stages of all the Pareto-optimal architectures.
3. The popular 3×3 convolution operator is never used in any optimal architecture.
4. The Pareto-optimal models share a lot of equal parameters: on average, 10 of 14 parameters are shared in the optimal models (while only four parameters are equal to the baseline model).
5. The NAS procedure prefers deeper models to our baseline. No models shallower than $3 + 2$ conv. stages exist in the Pareto Set.
6. Only one Pareto-optimal architecture removed the concurrent convolutions in the convolutional building-block.

5.2 | Other unsuccessful attempts

We have done several experiments with no better results: using the parametric rectified unit (PReLU) activation function, adding the drop-out regularization, join-using global-average-pooling values aside global-max-pooling, using one or two dense layer(s) in the final stage, and altering the α factor. Severe image degradation for data augmentation resulted in poor convergence and high error rates (even with doubling the training epochs).

5.3 | Overall key findings

Notable findings of our studies in NAS for the aforementioned template matching application are:

1. Lightweight CNNs can achieve good results in small object detection, with computational load comparable to template-matching techniques. (A good lightweight CNN may use a small number of filters, e.g., less than 8, while having enough deepness.)
2. We have proposed a general CNN hyper-architecture capable of being used in many object detection applications.
3. NAS on lightweight CNNs is relatively robust to the hyper-architecture of the network.
4. Local-minima-avoidance strategy is necessary for Local search in NAS. Without it, the algorithm may trap in a nearby local-minima which is not suitable.

Further optimizations for our application rely on many techniques including network pruning, integer calculation, architecture search on the building-blocks, considering a wider variety of neural network blocks such as recurrent blocks, and fine-tuning the data augmentation process.

6 | CONCLUSIONS

In this paper, we reported a NAS procedure to achieve a fast and accurate template matching algorithm and for the evaluation, we applied it to the specific application of Licence Plate Detection. We proposed an architecture template and used our Iterated Local Search procedure, reaching Pareto-Optimal architectures that provide various Error/Computational-Cost trade-offs. We provided studies on the optimal architectures.

Future works include building-block optimization, analysing the possibility of reusing the intermediate presentation in the character recognition stage, exploiting other primitive operations such as recurrent blocks, more flexible hyper-architecture, and newer layers (and even creative layers with zero or low computation effort). Considering hardware-accelerated operations based on the vector processors [such as the NEONTM Single-Instruction Multiple-Data (SIMD) operations of the ARM processors] in the NAS operation may further improve the speed of the implementation. Also, a wide number of analyses could be performed, such as robustness to adversarial examples as well as other degradations such as the blur, domain drift and domain shift analysis, among others.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are openly available in github at https://github.com/shariatzadeh/IranianLPD-NAS.

ORCID

Mahmood Fathy https://orcid.org/0000-0003-0852-5488

REFERENCES

Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). *Yolov4: Optimal speed and accuracy of object detection. arXiv preprint arXiv:2004.10934*.

Bulan, O., Kozitsky, V., Ramesh, P., & Shreve, M. (2017). Segmentationand annotation-free license plate recognition with deep localization and failure identification. *IEEE Transactions on Intelligent Transportation Systems*, *18*(9), 2351–2363.

Chollet, F. (2017a). Xception: Deep learning with depthwise separable convolutions. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 1251–1258).

Chollet, F. (2017b). Xception: Deep learning with depthwise separable convolutions. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 1800–1807).

Elsken, T., Metzen, J. H., & Hutter, F. (2019). Neural architecture search: A survey. *Journal of Machine Learning Research*, *20*(55), 1–21.

Jaderberg, M., Simonyan, K., Zisserman, A., & Kavukcuoglu, K. (2015). Spatial transformer networks. *Proceedings of the 28th International Conference on Neural Information Processing Systems* (pp. 2017–2025).

Jin, H., Song, Q., & Hu, X. (2019). Auto-keras: An efficient neural architecture search system. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (pp. 1946–1956).

Liang, T., Glossner, J., Wang, L., Shi, S., & Zhang, X. (2021). Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, *461*, 370–403.

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., & Murphy, K. (2018). Progressive neural architecture search. *Proceedings of the European Conference on Computer Vision (ECCV)* (pp. 19–34).

Liu, H., Simonyan, K., & Yang, Y. (2018). Darts: Differentiable architecture search. *International Conference on Learning Representations*.

Pham, H., Guan, M., Zoph, B., Le, Q., & Dean, J. (2018). Efficient neural architecture search via parameters sharing. *International Conference on Machine Learning* (pp. 4095–4104). PMLR.

Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement.

Springenberg, J. T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2014). Striving for simplicity: The all convolutional net. *arXiv Preprint arXiv:1412.6806*.

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., & Le, Q. V. (2019). Mnasnet: Platform-aware neural architecture search for mobile. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 2820–2828).

Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *Proceedings of the* 2001 *IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, IEEE.

Weihong, W., & Jiaoyang, T. (2020). Research on license plate recognition algorithms based on deep learning in complex environment. *IEEE Access*, *8*, 661–675.

Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 8697–8710.

AUTHOR BIOGRAPHIES

Seyed Mahdi Shariatzadeh received the B.Sc. degree in computer engineering from Iran University of Science and Technology and the M.Sc. degree in computer science from Sharif University of Technology, Iran, in 2008 and 2011, respectively. He is currently a Ph.D. candidate at the IUST, focused on neural architecture search and deep convolutional neural networks. His research interests include deep learning, CNN architectures, and computer vision.

Mahmood Fathy received B. S in electronics from Iran university of science and technology in 1984. He received M.S. in microprocessor engineering from Bradford university in 1986 and Ph. D in computer architecture and image processing from Manchester university in 1991. He is currently a professor in computer engineering department. His research interests are artificial intelligence, computer vision, machine learning, and deep learning.

Reza Berangi received the B.S. in Telecommunication Engineering and M.S. in Electronic Engineering from Iran University of Science and Technology, Tehran, Iran, in 1985 and 1989, respectively. He received Ph. D in Mobile Telecommunications from Victoria University of Technology, Melbourne, Australia, 1998. He is currently a professor in computer engineering department specialized in telecommunications. His research interests are artificial intelligence, deep learning and information technology.

How to cite this article: Shariatzadeh, S. M., Fathy, M., & Berangi, R. (2023). Improving the accuracy and speed of fast template-matching algorithms by neural architecture search. *Expert Systems*, *40*(9), e13358. https://doi.org/10.1111/exsy.13358