Design and deployment of an efficient landing pad detector

Andrea Albanese[2⋆], Tommaso Taccioli[1⋆], Tommaso Apicella[1], Davide Brunelli[2], and Edoardo Ragusa[1]

[1] Electrical, Electronics and Telecommunication Engineering and Naval Architecture Department (DITEN), University of Genoa, Genoa, Italy
{4398901@studenti,tommaso.apicella@edu,edoardo.ragusa@edu}.unige.it
[2] Department of Industrial Engineering (DII), University of Trento, Trento, Italy
{andrea.albanese, davide.brunelli}@unitn.it

Abstract. Robust landing pad detection plays a major role in Autonomous Unmanned Aerial Vehicles (UAVs). This problem can be approached using deep neural networks for vision-based inference. However, the full integration of deep learning algorithms into the small UAVs is still challenging for their limited resources. This paper presents a landing pad detection pipeline based on a revisited version MobileNetV3-Small. The proposed architecture inherits robustness from the general-purpose version but limits the computational cost significantly thanks to a set of design criteria aimed to limit hardware requirements. Experimental results confirm that the proposed network compares favorably with a lightweight general-purpose object detector in terms of accuracy/computational cost trade-off. The system is also deployed on a commercial general-purpose microcomputer confirming that satisfactory performance can be obtained on general-purpose embedded architectures.

Keywords: CNNs · Embedded systems · Autonomous Unmanned Aerial Vehicles · Edge computing.

1 Introduction

The successful application of Autonomous Unmanned Aerial Vehicles (UAVs) necessarily passes through the solution of complex problems of computer vision (CV). Although Deep learning (DL) proves to be the state-of-the-art for CV, resource-constrained devices struggle in supporting DL solutions in real-time. Eventually, this becomes a major limitation for the development of UAVs.

This paper presents a lightweight deep neural network (DNN) that can detect landing pads in aerial images. The proposed vision-based landing detector consists of a DNN based on a simplified version of MobileNetV3-Small [1]. As merit, the proposal limits the computational impact considering all the peculiarities of the problem under analysis. The intuition derived from the analysis

⋆ The two authors equally contributed to the Manuscript.

of the problem was combined with a layer ablation study that confirmed the suitability of the proposed changes in the neural architecture.

As a major result, this solution enables autonomous control of small UAVs by processing the task directly in embedded systems mounted in the drones. The autonomous inference enables the UAV to detect a safe landing area and navigate accordingly to land on it.

As a proof of concept, the network was deployed in the Raspberry Pi 4 model B Single Board Computer (RPi) to evaluate its performance with an edge computing device. RPi sets a challenging benchmark because is not designed to host DL inference, but it is commonly used as a computing platform on small-size drones.

The approach was validated using a real-world dataset of aerial images. Results confirmed that the proposed network compares favorably with a recently proposed object detector, especially in terms of computing cost. In addition, the measure of energy consumption and the inference time confirm the suitability of the proposed approach.

The contribution of the paper can be summarized as follows:

- The paper presents a detailed analysis that leads toward the application of a deep learning pipeline for robust landing pad detection;
- The ablation study provides insight into the computer vision problem that can be employed in future studies;
- The deployment on a RPi confirms that the lightweight network can be supported by commercial devices suitable for the target application.

2 Related works

Small UAVs may feature limited computational and energy resources, challenging autonomous navigation. So far, visual-based navigation is the most used configuration because low-power computer vision systems are becoming more and more accessible [2, 3]. Researchers have recently proposed combining vision-based systems with Machine Learning (ML) algorithms to bring intelligence into UAV's missions. [4] combines traditional computer vision algorithm with DNN to develop a safe landing area detection system. This combination allows the preliminary process of the video to extract relevant features and then feed the DNN with the preprocessed video enhancing system efficiency. This is a successful approach which enhances the system's performance and energy efficiency. [5] proposes a two-stage training system to detect landing areas. A CNN trained over synthetic images is used in the first stage, then a custom Kalman Filter controller is used to have an accurate control during landing and approach the landing site. The second stage consists of a transfer learning approach that uses weights produced by the first stage CNN. With this two-stage system, it is possible to have a flexible model that can be employed in different scenarios. In general, ML algorithms can be used in mobile robotics applications adopting one of the two main approaches: cloud computing and edge computing.

Cloud computing consists of transmitting sensed data to a server with near-unlimited computational resources to compute the prediction using a ML model and send the result back. However, data transmission introduces a considerable latency and energy consumption. For instance, [6] presents an object detection system for UAVs by using Region-based Convolutional Neural Networks (R-CNN) and the cloud computing approach. "Dronetrack" [7] is another system developed for UAVs that use cloud-based real-time object tracking. This paper revealed that fast connections such as optic fiber lead to real-time execution. These results suggest that running ML applications with a cloud computing approach is feasible. However, these systems are network-dependent. Thus, if it is considered an application that can be scaled in different scenarios where it is not always possible to have a stable wireless connection, the cloud computing approach is not the best choice.

The edge computing approach overcomes the cloud limitations leading to better execution performance. For this reason the most disparate applications were refined to be hosted in edge devices, including but not limited to semi-autonomous prosthesis [8], pest detection [9, 10], visual sentiment analysis [11]. For instance, [12] presents a system for perception, guidance, and navigation for racing drones. This system uses DL techniques to estimate the racing gate centers and assist drones in navigating through gates. It uses NVIDIA Jetson TX2 as a computer on-board responsible for computing the prediction in real-time, achieving a frame rate of 28.95 FPS. The same board is used in [13] for enabling real-time object detection in a UAV warning system. It exploits the state-of-the-art YOLO V2 [14] for recognizing dangerous products with visual aerial inspection directly on the drone board. Platforms such as GPUs and SBCs can accommodate complex models requiring a high memory footprint, while MCUs can host hundreds of KBs. Thus, the choice of the target platform is conditioned to the model that has to host. For instance, [15] use Raspberry SBCs as target platform to enable autonomous navigation on UAVs. The used CNNs are tailored to produce fast results with the computational power of Raspberry and with a low-power profile. On the other hand, if it is necessary to use huge ML models with resource-constrained platforms (e.g., MCUs), optimization techniques become necessary to fit ML models in the target platform.

Aside from platforms development, a surge of dedicated DL architectures were proposed in recent years to enable edge computing. The most used optimization techniques for predefined architectures are pruning [16], weights compression [17], quantization [18], and low-rank approximation [19]. However, the starting architecture significantly bias memory footprint, run-time memory requirements, and the number of floating point operations performed during inference phase (FLOPs). In recent years, researchers have proposed new operations that decrease computation requirements: depthwise convolution (DWConv) [20], mobile inverted bottleneck (MBConv) [21], efficient activation [22]. The latest research trends for lightweights architecture definition were focused on knowledge distillation [23] and Neural Architecture Search (NAS) [24]. In practice, most of the latest general purpose CNN for computer vision such as MobileNetV3 [25]

and EfficientNet [26] were obtained using NAS-based procedure. However, NAS-based approaches induce a computational overload in the training phase, which requires one to rely on dedicated computing equipment. In addition, the constraints introduced by a device such as a low-cost microcontroller would heavily impact the design of the eventual loss function, thus affecting the optimization phase.

3 Proposal: Design of a DNN for landing detection

MobileNetV3-Small architecture was proposed as a lightweight general-purpose deep network [1]. The network is built on top of a core building block called Bottleneck. The original paper uses the backbone network to target two different applications: object detection and semantic segmentation. Semantic segmentation provides a bit-wise classification of the pixels in the form of masks. Instead, object detection produces a number of bounding boxes, each one associated with a class and confidence. In some cases, DNNs for semantic segmentation feature fewer parameters than object detection alternatives. In our case, the two solutions presented in MobileNetV3 paper featured 0.47 M parameters for segmentation and 1.77 M parameters for the detector. Given the smaller footprint, we propose to use a customized version of the semantic segmentation architecture for landing pad detection. In the last part of the section, we present the low-cost strategy that converts the mask into the coordinates of the landing pad.

Even being one of the smallest DNNs for image classification MobileNetV3 is largely oversized for the target problem. This architecture was designed to extract high-level semantic features; meanwhile, the target problem involves the detection of target objects with simple geometric patterns. For this reason, large parts of the network are useless and inflate the compute cost. It is well-known that the higher layer of the networks extracts high-level semantic features. In the original implementation, the 9th bottleneck block and the last 2D convolution fed the segmentation head [1]. We propose the 2nd and 5th bottleneck blocks as inputs to the segmentation head. This means that the layers after the 5th bottleneck block have been excluded from the architecture, thus improving execution speed and reducing the total number of parameters. Among the candidates, the 2nd and 5th bottleneck blocks were selected to trade-off generalization capabilities and compute cost because the size of the output tensors elaborated by these two blocks matched the input size required by the segmentation head.

In addition, the architecture has been tailored to handle small resolution images. The original design of MobileNet-V3 supports input with 1024×2048 pixels resolution. High-resolution images are not required in landing pad detection, since it could be sufficient to distinguish the landing pad from the background, and introduce an increment in resources utilization. Furthermore, the original architecture uses stride different from 1 in the first level of the architecture that can negatively affect the landing pad's detection when distant. We tackled this issue by setting the stride of the first convolution layer and second bottleneck layer from 2 to 1. The stride acts as a compression factor on the image size.

Table 1. Proposed Backbone. KEY - Operator: building block; Exp size: expansion factor; Out: number of output filters; SE: Squeeze-And-Excite ; NL: nonlinear activation; RE: ReLU; HS: hard swish; s: stride.

Input	Operator	Expsize	Out	SE	NL	s
$320^2 \times 3$	conv2d,3×3	-	16	-	HS	1
$320^2 \times 16$	bneck,3×3	16	16	v(1)	RE	2
$160^2 \times 16$	bneck,3×3	72	24	-	RE	1
$160^2 \times 24$	bneck,3×3	88	24	-	RE	1
$160^2 \times 24$	bneck,5×5	96	40	v(2)	HS	2
$80^2 \times 40$	bneck,5×5	240	40	v(3)	HS	1

Eventually, the new version of the network envisions an input size of $[320 \times 320]$. The final output of the designed DNN is a heatmap H of size 160 x 160 with a probability value that corresponds to the probability of the pixel belonging to a landing pad. Table 1 summarizes the architecture of the proposed backbone network highlighting the most import parameter on the columns.

Squeeze-And-Excite (SE) layer plays a major role in the original definition of MobileNetV3, both in terms of generalization performance and compute cost. On the one hand, these blocks contain fully connected layers that contribute significantly to the final number of parameters. On the other hand, SE effectively propagates the information toward different layers of the network. Given that we used a small portion of the original network (around half of the original depth), we can speculate that the contribution of SE layers will be negligible. Eventually, this operation reduced the memory requirements significantly of the network with a negligible impact on the performance of the target problem. An ablation study will confirm this claim in the experimental setup.

The lightweight pipeline in Figure 1 performs all the tasks that convert the output of the segmentation network into the 2D coordinates of the landing pad. First, a threshold operation filter out the pixel with a probability lower than a reference value. Then, the well-known algorithm described in [27] groups the pixel with a significant probability in clusters, as shown in the figure. Finally, the cluster having the largest perimeter is selected as the predicted box.

4 Experiments

4.1 Generalization performance validation

A dataset composed of 13 videos of landing pads containing a total of 17,720 frames has been collected. The videos are collected from three different heights: approximately from 3, 4 and 6 meters. In each frame one out of three landing pad types are present. Figure 2 reports three examples of images from the dataset with additional zero-padding. 5,542 frames from 5 of these videos have been employed as training sets. The remaining images from the same videos have been then discarded, leaving 11,300 images for the test. The experiments involved two distinct test sets. A first set, called test1, contains all the testing images.

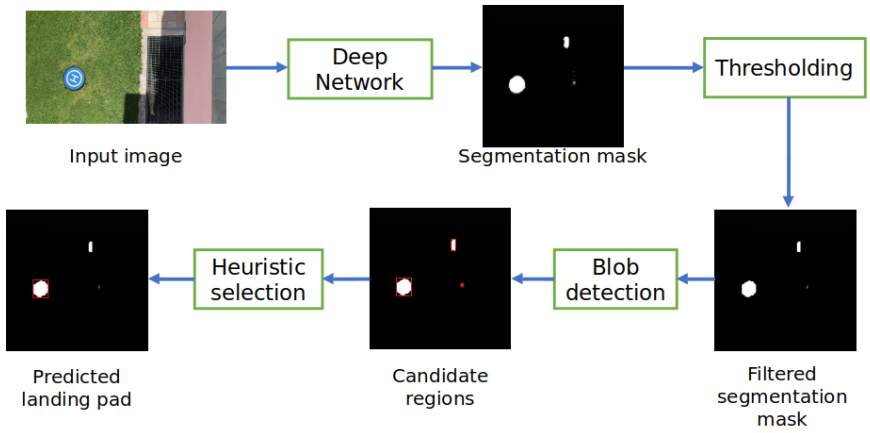

Fig. 1. Pipeline from the RGB input to the landing pad detection.

A subsection of this testing set, called test2, corresponding to 3,328 images belonging to 2 videos never included in training operations, has been employed as a more challenging test set. We obtained the masks using a weak labeling strategy. A MobileNetV3-SSD (Small) pretrained on COCO dataset for object detection has been fine-tuned on a subset of the dataset that has been manually labeled. This network has been used to annotate the remaining images. The eventual bounding boxes have been further refined using a thresholding strategy based on the color of the landing pad. It should be noted that this step was a semi-manual annotation and cannot be included in a fully autonomous pipeline.

Visual inspection from two human users confirmed that the prediction obtained were sufficiently overlapped with the landing pad. Eventually, the MobileNetV3 - SSD sets the comparison in the experimental setup.

Fig. 2. Example of landing pad images contained in the dataset.

An ablation study confirmed the hypothesis about the contribution of the SE layers. The network was trained for 10 epochs, with batch size 4, learning rate 0.0001. Table 2 summarizes the results of the ablation study, showing the performance of the network in terms of Intersection over Union (IoU). The IoU, a.k.a. Jaccard index, is a measure of how much the predicted bounding box (A) is comparable with respect to the ground truth (B) following the relationship: $IoU = \frac{|A \cap B|}{|A \cup B|}$. We compared 5 configurations, Full refers to the network as presented in table 1, *SE(n) excluded* identifies the network without the n-th SE layer. The second and third columns show the number of parameters of the complete architecture and the number of floating-point operations for one inference phase, respectively. Column IoU_{tot} reports the average IoU value measured between the labels and the weak labels. Errors report the number of images where the IoU value was 0. Finally, column IoU_{clean} reports the average IoU only for images where the measured IoU was greater than 0. The performance was measured using the whole test set, i.e., test1.

Table 2. Ablation study results. KEY - Params: Number of parameters; FLOPs: Number of floating-point operations; G: 10^9; IoU_{tot}: average Intersection over Union; IoU_{clean}: average of Intersection over Union greater than 0.

Architecture	**Params**	**FLOPs (G)**	**IoU$_{tot}$**	**Errors**	**IoU$_{clean}$**
Full	195,460	1.295	71.6%	1,093 (9.7%)	79.3%
SE(1) excl.	194,916	1.294	77.5%	31 (0.3%)	77.7%
SE(2) excl.	176,836	1.293	76.5%	31 (0.3%)	76.7%
SE(3) excl.	79,780	1.291	69.0%	403 (3.6%)	71.6%
SE(1,2,3) excl.	60,612	1.289	72.8%	530 (4.7%)	76.4%

The results highlight that SE blocks play a major role in the number of parameters. In particular, SE(3) has a huge impact on the number of parameters. The number of operations is heavily affected by these layers, but the relative contribution is smaller because dense layers impact weights but not significantly on tensor size propagation. In terms of generalization error, we can note that SE blocks affect overall performance but do not always positively affect the network. The solution without SE blocks proves more accurate than the version without the third SE when images with IoU 0 are excluded. The larger number of errors for the Full architecture was because, in some cases, masks were significantly noisy. A visual inspection confirmed that the prediction marked as an error was correct in a few cases.

Table 3 summarizes the comparison with the MobileNetV3-SSD (V3-SSD), i.e. the solution that would have been naturally employed from [1]. The table reports number of parameters (in millions) and FLOPs measured in Giga (G) followed by the average IoU computed over the challenging test set, i.e., test2. IoU columns report, in addition, between brackets, the average IoU considering only images where IoU was greater than 0.

Table 3. Comparison with state-of-the-art model for lightweight object detection. KEY - Params: Number of parameters; M: 10^6; FLOPs: number of floating-point operations; G: 10^9; IoU: average Intersection over Union.

Architecture	Params (M)	FLOPs (G)	IoU
V3-SSD	2.761	3.758	69.3%
Proposal	0.061	1.289	61.7% (63.7%)

The results highlight that the proposed approach includes 45 times fewer parameters and requires around one-third of operations with respect to the original object detector. When analyzing the IoU, one must consider that those analyses in biased towards the SSD. In fact, this model has been used to build most of the labels. Accordingly, the 8% gap in terms of the IoU can be ascribed to noise introduced by the weak labeling process.

4.2 Model optimization

The developed system for landing pad detection was deployed on the RPi board. The system was characterized by considering energy consumption, frame rate, and memory usage for processing one frame by averaging the measured values. Moreover, the system performance is compared among four model configurations taking into account the architecture presented in Section 3 with INT8, FP16, and FP32 bit representation, along with MobileNetV2 (MobV2) based setup used in [15] for landing pad detection with FP32 bit representation. The measurement campaign was conducted by processing about 700 frames for each configuration in a static scene. Specifically, the system was mounted in order to view and process a landing pad at a distance of three meters continuously. Moreover, a General Purpose Input Output (GPIO) was configured to give an impulse of 100 ms at the end of each frame to synchronize the processed frames with the relative current consumption. An oscilloscope was used to monitor the current consumption as well as the GPIO with a sampling frequency of 50 kHz. The measurements of current consumption relative to the processing time were analyzed to find the average time and the average current consumption. The results of the overall system characterization are summarized in Table 4.

As shown in Table 4, the 32 bit configuration is the most efficient and with the best frame rate. The 8 bit configuration, indeed, shows the worst performance because it is not supported by the RPi's instruction set. Moreover, MobileNetV2 needs an additional piece of pipeline to solve the problem of landing pad detection, thus confirming the efficiency of the proposed solution. The overall system characterization for the best configuration (FP32) reveals that the average processing of one frame takes 388.05 ms and consumes 1.80 J. Moreover, it uses only 2.27% of the available RAM (i.e., 4 GB), achieving 2.58 Frames Per Second (considering a the average time to perform the processing), enough to compute a landing trajectory and correct eventual drifts. The presented system runs onboard on UAVs with the edge computing approach. Considering a small-size UAV, it is equipped with a 1500 mAh battery operating at 14.8 V which ensures

Table 4. Comparison between the proposed model and MobileNetV2 [15] in terms of energy consumption, execution time, measurement standard deviation and memory usage, varying the quantization. KEY - Config: quantization used; INT8: 8-bit integers; FP16: Floating-point 16 bits; FP32: Floating-point 32 bits; MobV2: MobileNetV2 quantized FP32; std: Energy standard deviation; Time: average time; FPS: Frames Per Second considering the average time; RAM: Random-access memory.

Config	Energy (J)	std	Time (ms)	FPS	RAM	Disk usage
INT8	2.22	0.0051	523.51	1.91	97.05 MB	108.17 kB
FP16	1.83	0.012	390.31	2.56	92.69 MB	132.98 kB
FP32	**1.80**	0.012	388.05	**2.58**	92.82 MB	766.31 kB
MobV2	1.80	0.0097	478.80	2.09	223.01 MB	26.70 MB

a flight time of 15 minutes. With the proposed system operating onboard, the drone's available energy and flight time are reduced by only 5%. This means that bringing intelligence to UAV systems does not compromise the available energy, which is a precious resource in such systems.

5 Conclusions

This paper provides a detailed analysis that leads toward the application of a deep learning pipeline for robust landing pad detection. The analysis is carried out through an ablation study that considers insights of the analysed computer vision problem. The eventual deployment on a RPi confirms that the lightweight network can be supported by commercial devices suitable for the target application. As future work, the integration of the RPi module with hardware accelerators e.g. Intel Neural Compute Stick, will provide additional boost to the performance of the network and additional clues concerning the real-time performance of the pipeline. Moreover, the employment of NAS techniques will lead to the development of an even more efficient DL model, achieving better performance, extending this solution in other UAVs autonomous navigation applications.

References

1. A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, et al., Searching for mobilenetv3, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 1314–1324.
2. J. Courbon, Y. Mezouar, N. Guénard, P. Martinet, Vision-based navigation of unmanned aerial vehicles, Control Engineering Practice 18 (7) (2010) 789–799.
3. M. H. F. M. Fauadi, S. Akmal, M. M. Ali, N. I. Anuar, S. Ramlan, A. Z. M. Noor, N. Awang, Intelligent vision-based navigation system for mobile robot: A technological review, Periodicals of Engineering and Natural Sciences 6 (2) (2018) 47–57.
4. M. Paszkuta, J. Rosner, D. Peszor, M. Szender, M. Wojciechowska, K. Wojciechowski, J. P. Nowacki, Uav on-board emergency safe landing spot detection system combining classical and deep learning-based segmentation methods, in: Asian

Conference on Intelligent Information and Database Systems, Springer, 2021, pp. 467–478.
5. P. Mathur, Y. Jangir, N. Goveas, A generalized kalman filter augmented deep-learning based approach for autonomous landing in mavs, in: 2021 International Symposium of Asian Control Association on Intelligent Robotics and Industrial Automation (IRIA), IEEE, 2021, pp. 1–6.
6. J. Lee, J. Wang, D. Crandall, S. Šabanović, G. Fox, Real-time, cloud-based object detection for unmanned aerial vehicles, in: 2017 First IEEE International Conference on Robotic Computing (IRC), IEEE, 2017, pp. 36–43.
7. A. Koubâa, B. Qureshi, Dronetrack: Cloud-based real-time object tracking using unmanned aerial vehicles over the internet, IEEE Access 6 (2018) 13810–13824.
8. E. Ragusa, C. Gianoglio, S. Dosen, P. Gastaldo, Hardware-aware affordance detection for application in portable embedded systems, IEEE Access 9 (2021) 123178–123193.
9. A. Segalla, G. Fiacco, L. Tramarin, M. Nardello, D. Brunelli, Neural networks for pest detection in precision agriculture, in: 2020 IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor), IEEE, 2020, pp. 7–12.
10. A. Albanese, M. Nardello, D. Brunelli, Automated pest detection with dnn on the edge for precision agriculture, IEEE Journal on Emerging and Selected Topics in Circuits and Systems 11 (3) (2021) 458–467.
11. E. Ragusa, T. Apicella, C. Gianoglio, R. Zunino, P. Gastaldo, Design and deployment of an image polarity detector with visual attention, Cognitive Computation 14 (1) (2022) 261–273.
12. S. Jung, S. Hwang, H. Shin, D. H. Shim, Perception, guidance, and navigation for indoor autonomous drone racing using deep learning, IEEE Robotics and Automation Letters 3 (3) (2018) 2539–2544.
13. N. Tijtgat, W. Van Ranst, T. Goedeme, B. Volckaert, F. De Turck, Embedded real-time object detection for a uav warning system, in: Proceedings of the IEEE international conference on computer vision workshops, 2017, pp. 2110–2118.
14. J. Redmon, A. Farhadi, Yolo9000: better, faster, stronger, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 7263–7271.
15. A. Albanese, M. Nardello, D. Brunelli, Low-power deep learning edge computing platform for resource constrained lightweight compact uavs, Sustainable Computing: Informatics and Systems (2022) 100725.
16. S. Ye, X. Feng, T. Zhang, X. Ma, S. Lin, Z. Li, K. Xu, W. Wen, S. Liu, J. Tang, et al., Progressive dnn compression: A key to achieve ultra-high weight pruning and quantization rates using admm, arXiv preprint arXiv:1903.09769 (2019).
17. E.-H. Yang, H. Amer, Y. Jiang, Compression helps deep learning in image classification, Entropy 23 (7) (2021) 881.
18. E. Park, J. Ahn, S. Yoo, Weighted-entropy-based quantization for deep neural networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 5456–5464.
19. Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, D. Shin, Compression of deep convolutional neural networks for fast and low power mobile applications, arXiv preprint arXiv:1511.06530 (2015).
20. F. Chollet, Xception: Deep learning with depthwise separable convolutions, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 1251–1258.
21. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 4510–4520.

22. P. Ramachandran, B. Zoph, Q. V. Le, Searching for activation functions, arXiv preprint arXiv:1710.05941 (2017).
23. J. Gou, B. Yu, S. J. Maybank, D. Tao, Knowledge distillation: A survey, International Journal of Computer Vision 129 (6) (2021) 1789–1819.
24. L. L. Zhang, Y. Yang, Y. Jiang, W. Zhu, Y. Liu, Fast hardware-aware neural architecture search, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, 2020, pp. 692–693.
25. A. Howard, M. Sandler, G. Chu, L. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, H. Adam, Searching for mobilenetv3, CoRR abs/1905.02244 (2019). arXiv:1905.02244.
URL http://arxiv.org/abs/1905.02244
26. M. Tan, Q. Le, Efficientnet: Rethinking model scaling for convolutional neural networks, in: International Conference on Machine Learning, PMLR, 2019, pp. 6105–6114.
27. S. Suzuki, et al., Topological structural analysis of digitized binary images by border following, Computer vision, graphics, and image processing 30 (1) (1985) 32–46.