Constrained evolutionary piecemeal training to design convolutional neural networks

Sapra, D.; Pimentel, A.D.

DOI
10.1007/978-3-030-55789-8_61

Publication date
2020

Document Version
Final published version

Published in
Trends in Artificial Intelligence Theory and Applications : Artificial Intelligence Practices

Constrained Evolutionary Piecemeal Training to Design Convolutional Neural Networks

Dolly Sapra(✉) and Andy D. Pimentel

University of Amsterdam, Amsterdam, Netherlands
{d.sapra,a.d.pimentel}@uva.nl

Abstract. Neural Architecture Search (NAS), which automates the discovery of efficient neural networks, has demonstrated substantial potential in achieving state of the art performance in a variety of domains such as image classification and language understanding. In most NAS techniques, training of a neural network is considered a separate task or a performance estimation strategy to perform the architecture search. We demonstrate that network architecture and its coefficients can be learned together by unifying concepts of evolutionary search within a population based traditional training process. The consolidation is realised by cleaving the training process into pieces and then put back together in combination with evolution based architecture search operators. We show the competence and versatility of this concept by using datasets from two different domains, CIFAR-10 for image classification and PAMAP2 for human activity recognition. The search is constrained using minimum and maximum bounds on architecture parameters to restrict the size of neural network from becoming too large. Beginning the search from random untrained models, it achieves a fully trained model with a competent architecture, reaching an accuracy of 92.5% and 94.36% on CIFAR-10 and PAMAP2 respectively.

Keywords: Neural networks · Neural architecture search · AutoML · Constraint optimization

1 Introduction

Recent work to discover efficient neural networks automatically has proven to be a highly efficient methodology, where the discovered neural network architectures are outperforming the hand-crafted ones. Popular approaches for Neural Architecture Search (NAS) use reinforcement learning [26,37] and evolutionary algorithms [23,27]. However they take tens to thousands of GPU days to prepare, search and converge. Most of these methods rely on resource heavy training to guide the search process. In this paper, we look at NAS from a different perspective by exploring the possibility of finding optimal architectures during the training process itself as opposed to accuracy prediction or training as a separate performance estimation strategy.

H. Fujita et al. (Eds.): IEA/AIE 2020, LNAI 12144, pp. 709–721, 2020.
https://doi.org/10.1007/978-3-030-55789-8_61

Moreover, most of recent work in NAS techniques is focused on image classification tasks such as for CIFAR-10 [18] and Imagenet [11] leading to innovative research on complex search spaces that are highly suited to vision tasks. These search spaces are derived from hand-crafted previous state-of-the-art such as residual connections [16], cell based designs such as inception [32], dense net [17] or generated in a graph-like fashion [31]. While these search spaces have proven to be extremely efficient, these are not always easy to understand, train and implement. Most medium complexity tasks and domains such as human activity recognition [33], earth sciences [20,25] and astronomical studies [9] deploy plain convolutional networks as they are considered sufficient as well as easy to understand by scientists from non-AI background. A tool to find an efficient Convolutional Neural Network (CNN) in a few GPU days and to be usable in any domain by non-AI experts is also a step forward in simplifying and democratizing AI. In this paper we look at the Human Activity Recognition (HAR) domain with the PAMAP2 [29] dataset where inputs from multiple body worn sensors are used to predict the activity being undertaken by the wearer. We also demonstrate versatility of our approach using a very different CIFAR-10 dataset for image classification.

The main contribution of this paper is a novel NAS algorithm that searches for an efficient CNN architecture for a given task and converges in a few GPU hours. Our work leverages a population based computing technique which allows a group of CNNs to train in parallel. During this training, evolutionary operators are applied to some random CNNs at regular intervals which leads to architecture modification and hence exploration of the search space. A new architecture derived like this is always partially trained already as it was modified from another architecture undergoing training. In subsequent iterations derived architectures continue to train. Towards the end of this algorithm, the best candidates are selected from the population, which can then be post processed or trained further, if needed. All the CNNs generated and modified during search are bound by minimum and maximum values for each architecture parameter. These constraints are in place to make sure that architectures do not become too big and limits the resource consumption of the final neural network. This is an important factor to consider for tasks intended to be used on embedded systems like wearables in HAR tasks.

The rest of the paper is organized as follows. We discuss related published works for NAS in Sect. 2. We then present methodologies and detail our algorithm in Sect. 3 and describe experimental setup and results of our evaluations and validations in Sect. 4. Finally, we conclude the paper in Sect. 5.

2 Related Work

There are many published works in the field of NAS, roughly divided into three groups - reinforcement learning based, evolutionary and one-shot architecture search. In reinforcement learning (RL) and evolutionary search methods, training the neural network completely with each iteration is mandatory to evaluate

its performance. In RL methods [4,26,37], the reward of the RL agent is dependent on the validation performance of the trained architecture. By continuously rewarding the agent the search is guided towards better performing neural networks. RL based approaches require the construction of an appropriate agent, which often itself is another neural network and its optimization also requires considerable effort in designing and fine tuning.

Evolutionary approaches [8,22,27,28,35] use genetic algorithms to optimize the neural architecture. These are population based algorithms where every iteration trains and evaluates all the architectures in the population. The subsequent generation of neural networks is chosen based on their prediction accuracy and average performance of the whole population increases with time leading to discovery of a highly efficient neural network architecture at convergence.

Unfortunately, a lot of these approaches require large computational resources as they need to train and validate hundreds to thousands of architectures. RL method [37] trained over 10,000 of neural architectures, requiring thousands of GPU days and another very efficient evolutionary search [13] still takes 56 GPU days to converge. Some works use proxy tasks and helpers such as hyper-networks [5], predictors [4,10] and controllers [26] to speed up the search, but even so they need significant preparation and construction time prior to the actual search. Our approach converges in a few GPU hours to a couple of GPU days without using any additional helper or proxy task.

One-Shot Architecture Search is another promising approach where NAS is modeled as a single training process of a super-network that comprises of all possible sub-networks. DropPath [38] drops out each path with some fixed probability and use the pre-trained super-network to evaluate architectures, which are sub-networks created by randomly zeroing out paths. DARTS [21] introduces an architecture parameter in addition for each path and jointly train weight parameters and architecture parameters using standard gradient descent. Other approaches might need to utilize proxy tasks to be viable such as [6] employs a memory-efficient scheme where only few paths are updated during search. The approaches which require the entire super-network to reside in GPU memory during NAS are restricted to relatively small architecture size, usually a cell that can be stacked to form multiple times to form the whole network. They usually also face a meta-architecture design problem after the search regarding number of cells to be used and how to connect them to build the actual model. Besides, a repeated single cell type may not be optimal for every application.

3 Methodology

We call our proposed approach Evolutionary Piecemeal Training, where piecemeal-training refers to training a neural network with a small 'data-piece' of size δ_k. A traditional continuous training is interceded by an evolutionary operator at regular intervals and the interval of intervention is dictated by the value of δ_k. An evolutionary operator modifies the architecture and the training further continues. This process is done for multiple neural networks forming a population and the candidates which are not able to achieve high accuracy during

training keep dropping out from the population. This can also be seen as early training termination of candidates that are performing poorly. In this section, we give details of the key concepts and then outline the complete Algorithm.

3.1 Search Space

The search space for our algorithm is focused on plain Convolutional Neural Networks (CNNs), which consist of convolutional, fully-connected and pooling layers without residual connections, branching etc. Batch normalization and non-linear activations are configurable and can be added to the network. CNN architectures are defined by a group of blocks, each block is of a specific layer type and is bounded by minimum and maximum number of layers it can have. Additionally, each layer has upper and lower bounds on each of its parameters. For example, a convolutional layer will have bounds on the number of units, kernel sizes and stride sizes possible. Section 4.2 defines the architecture search spaces for CIFAR-10 and PAMAP2 respectively. The search space specifications along with its bounds are encoded as a collection of genes, also called a genotype. All possible combinations of parameters together form the gene pool from which individual neural networks are created and trained.

3.2 Population Based Training

We employ a population based training process where an initial population of neural networks is randomly created from the defined gene pool. In each iteration, all candidates of the population are piecemeal-trained and then evaluated using the validation set. Depending upon the available resources, all candidates can be trained in any combination of parallel and sequential manner. The size of the population is kept constant throughout the algorithm, though the candidates of the population keep changing as they are altered through the evolutionary operators applied in each iteration. The number of candidates in the population needs to be large enough to maintain enough diversity of CNN architectures in the population, while still satisfying the constraints applied to it.

3.3 Evolutionary Operators

Evolutionary algorithms are iterative population based algorithms where we evolve a better population over subsequent iterations. During each evolutionary step, some candidates from the population of CNNs are altered once using evolutionary operators called recombination and mutation operators.

Recombination works with two neural networks and swaps all layers in a gene-block of the same type. In this replacement, the layers being swapped are roughly in the same phase of feature extraction. The input and output feature map sizes of layer block being swapped are also identical in both of the selected networks. Figure 1 illustrates the recombination operator for swapping convolutional layers from two networks. Recombination is not a function preserving operator, but

in the experiments they were found to be important to introduce diversity in the population by changing the total number of layers in a candidate through swapping. To reduce the negative effect of loss incurred due to recombination, we used a cooling-down approach to the recombination rate. In earlier iterations, where the training loss is already high, there are more swaps happening than in the later ones, where training loss is very low.

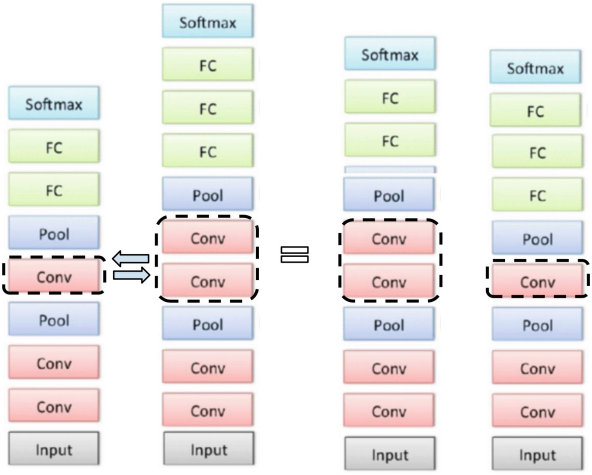


Fig. 1. Recombination operator on two neural networks swapping convolution layers

Mutation changes a layer's parameters such as number of kernels or kernel size and are designed to be function preserving. We apply the Net2Wider operator from [7] to increase the number of units and pruning [19] to reduce the number of filters in the layer. Filters are centrally cropped or zero-padded when their size is changed. Every mutation disrupts the ongoing training of the mutated candidates and some additional loss is incurred in the training-in-process. However, these specific operators were chosen because these are either completely or partially function preserving in nature, which means that the loss incurred from these operators is as small as possible and recoverable in later piecemeal-training.

3.4 Algorithm

Algorithm 1 outlines the complete algorithm, the goal is to find neural networks satisfying the architecture constraints with minimum prediction error.

$InitializePopulation()$ generates a neural network population of size N_p using a factory pattern class for the genotype and initializes them by training them for 1 epoch. Afterwards, this iterative algorithm runs for N_g generations. $PiecemealTrain()$ trains all individuals with randomly selected data of size δ_k

Algorithm 1: Evolutionary Piecemeal Training

Evolutionary Inputs: N_g, N_p, P_r, P_m, α
Training Inputs : τ_{params}, δ_k

$\wp_o \leftarrow InitializePopulation(N_p)$
for $i \leftarrow 0 \ldots N_g$ **do**
  $\wp_i \leftarrow PiecemealTrain(\wp_i, \tau_{params}, \delta_k)$
  $E_v \leftarrow ValidationPopulation(\wp_i)$
  $\wp_{best} \leftarrow BestSelection(\alpha, \wp_i, E_v)$
  $\wp_r \leftarrow random((1-\alpha) * \wp_i)$
  update $\wp_i \leftarrow \wp_{best} + \wp_r$
  $\wp_{rc} \leftarrow Recombine(\wp_i, P_r)$
  $\wp_{mu} \leftarrow Mutate(\wp_i, P_m)$
  update $\wp_i \leftarrow \wp_{mu} + \wp_{rc} + \wp_{remaining}$
end
return $BestCandidatesOf(\wp_{N_g})$

using τ_{params} training parameters. $ValidationPopulation()$ evaluates the population using the validation set and $BestSelection()$ selects α best individuals using the accuracy on validation set achieved so far. α is kept at very high ratio $> 0.95 * N_p$ so only a few candidates are rejected in every iteration. By doing this our focus is on rejecting poor performing architectures, as opposed to promoting an architecture that learns fast but might not be able to reach very high accuracy in the end. Population size, N_p, is kept constant by randomly selecting $1 - \alpha$ networks from survivors and added back to the pool. Random selection ensures that children generations are not overwhelmed by only one type of architecture which might have reached high accuracy by luck. $Recombine()$ and $Mutate()$ are evolutionary operators, they select individuals from the population with selection probability of P_m and P_r respectively. P_r is linearly cooled to ≈ 0 from its initial value. The algorithm returns the best candidates of CNNs from the final population. Best candidates can be further processed, modified or trained as needed outside this algorithm.

4 Experiments

In this section, we evaluate our algorithm using two datasets and outline the experimental setup. We present the datasets and data augmentation techniques used, their respective constrained search spaces and finally the results obtained. We have used the Java based Jenetics library [1] for evolutionary computation and the Python based Caffe2 [3] library for training and testing. We used the ONNX [2] format to represent and transfer the neural networks across different modules. Our experiments run on only one GPU (GeForce RTX 2080).

4.1 DataSets

For experiments, we use the CIFAR-10 dataset for image classification and the PAMAP2 dataset of human activity recognition. CIFAR-10 consists of 60,000 labeled images of dimensions $32 \times 32 \times 3$, comprising of 50,000 training and 10,000 testing images. The images are divided into 10 classes. We use 5,000 images from the training set as a validation set and we use only this validation set to guide the search process. The test set is used only in the end to check final accuracy of the network, which is what we report in this paper. We use standard data augmentation as described in [14,30] with small translations, cropping, rotations and horizontal flips. For piecemeal-training, δ_k random images are picked from the training set from which approximately 50% are augmented images.

The PAMAP2 dataset provides recordings from three Inertial Measurement Units (IMU) and a heart rate monitor. Together, the input data is in the form of time-series from 40 channels. In this dataset, nine subjects performed twelve different household, sports and daily living activities. We do not consider the optional activities performed by some subjects. Following [15], recordings from participants 5 and 6 are used as validation and testing sets respectively. IMUs' recordings are downsampled to 30 Hz and a sliding window approach with a window size of 3s (100 samples) and step size of 660ms (22 samples) is used to segment the sequences. To augment the data, a sliding window is moved by different step sizes while keeping the window size the same at 3s.

4.2 Search Space

Table 1 and Table 2 describe the search space specifications. Total design points for CIFAR-10 and PAMAP2 are to the order of 10^8 and 10^5 respectively. Number of units per layer can be multiples of 16 for CIFAR-10 and 8 for PAMAP2.

Table 1. Cifar-10 architecture search space

Layer type	Layers		Units/Layer		Kernel-size		Stride
	L_{min}	L_{max}	U_{min}	U_{max}	K_{min}	K_{max}	S_t
Convolution	2	5	48	96	3×3	7×7	1
MaxPool	1	1	1	1	2×2	2×2	2
Convolution	2	7	80	320	3×3	7×7	1
MaxPool	1	1	1	1	2×2	2×2	2
Convolution	2	7	256	640	3×3	7×7	1
MaxPool	1	1	1	1	2×2	2×2	2
Fully Connected	2	3	128	1024	–	–	–
SoftMax	1	1	1	1	–	–	–

Table 2. PAMAP2 architecture search space

Layer type	Layers		Units/Layer		Kernel-size		Stride
	L_{min}	L_{max}	U_{min}	U_{max}	K_{min}	K_{max}	S_t
Convolution	2	4	64	128	3×1	7×1	1
MaxPool	1	1	1	1	2×1	2×1	2
Convolution	2	5	96	256	3×1	7×1	1
GlobalMaxPool	1	1	1	1	2×1	2×1	2
Fully Connected	1	3	128	512	–	–	–
SoftMax	1	1	1	1	–	–	–

4.3 Training Setup

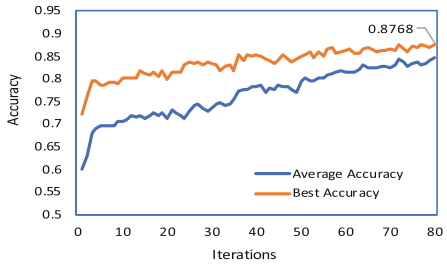
We trained the CIFAR-10 dataset with 80 generations and population size of 80. δ_k for piecemeal training is set to 4k random images. Each convolution and fully connected layer is followed by ReLu activation. Training was done using the Adam optimizer and batch size of 80 with initial learning rate of $5e^{-4}$ with step learning rate decay, after every 20 iterations learning rate was reduced by $1e^{-4}$. Both evolutionary selection probabilities P_m and P_r are initialized to 0.3. P_m stays constant, while P_r is linearly decayed to reach 0.01 at the last iteration.

PAMAP2 was trained for 30 generations and population size of 50. δ_k for piecemeal training is set to 20k random sensor samples. Each convolution and fully connected layer is followed by ReLu activation. Training was done using the Adam optimizer and batch size of 100 with initial learning rate of $1e^{-4}$ with constant learning rate. Evolutionary selection probabilities P_m and P_r are initialized to 0.3 and 0.3 respectively. As with the CIFAR-10 experiment, P_m stays constant and P_r is linearly decayed.

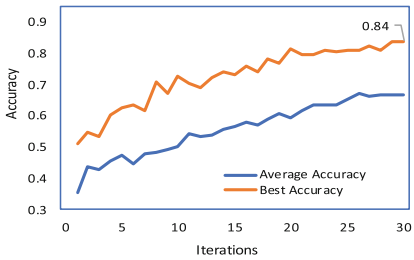
Neural networks for CIFAR-10 are larger than for PAMAP2, with a single GPU with 11 GB memory 4 parallel training threads for CIFAR-10 and 7 parallel training threads could run simultaneously. The limitation on the level of parallelism was defined by the available memory on the GPU. Also to fasten up the search there was no Batch Normalization used as it consumes more memory and reduces possible parallelism. After the search, the best candidate was modified and every convolutional layer was appended with a batch normalization layer and further trained for 100 epochs.

4.4 Results

In this section, we evaluate our algorithm and compare with other state-of-the-art. Figure 2 shows the training curves for experiments on CIFAR-10 and PAMAP2, with each iteration there is a general increase in average accuracy of the population as well as the best accuracy of an individual in the population. The best model might change from an iteration to the next. The best models



(a) Training curve for CIFAR-10



(b) Training curve for PAMAP2

Fig. 2. Training curves. Average Accuracy refers to the average performance of whole population at the given search iteration. Best accuracy refers to best found performance of an individual model from the population.

found at the end of all iterations were further trained to achieve final accuracy as reported in Fig. 2.

For CIFAR-10, the search took 2-GPU days and the best prediction accuracy was found to be 92.5% on the test set. Table 3 shows comparisons with other evolutionary approaches. We know that 92.5% is relatively low compared to other published works, but this is on a very simple and plain CNN without any architectural enhancements or advance data augmentation like mixup [36] or cutout [12]. Other approaches use a hybrid search space where different architecture blocks or cell modules as well as arbitrary skip connections are used. Instead of stacking conventional layers, these stack different blocks. The best model found in our experiments has 13 convolutional layers followed by 2 fully connected layers.

Table 3. CIFAR-10 accuracy comparisons with evolutionary approaches

Model	Search space	GPU-days	Accuracy (%)
CoDeepNeat [23]	Hybrid	–	92.7
GeneticCNN [35]	Hybrid	17	92.9
EANN-Net [8]	Hybrid	–	92.95
AmoebaNet [27]	Cell	3150	**96.6**
NSGANet [22]	Hybrid	8	96.15
Evolution [28]	Hybrid	1000+	94.6
EPT (ours)	Plain CNN	**2**	92.5

For the PAMAP2 dataset, we report the classification accuracy as well as the weighted F1-score ($F1_w$) and mean F1-score ($F1_m$), as this is an unbalanced set. These scores consider the correct classification of each class equally, using the

precision, recall and the proportion of class in the dataset. These are calculated as below:

$$F1_w = \sum_i 2 \times \frac{n_i}{N} \times \frac{precision_i \times recall_i}{precision_i + recall_i}$$

$$F1_m = \frac{2}{N} \times \sum_i \frac{precision_i \times recall_i}{precision_i + recall_i}$$

where, n_i is the number of samples for each class $k \in K$ and N is the total number of samples in the dataset. Compared with the classification accuracy, the F1-scores are more convenient for evaluating the performance of the networks on highly unbalanced datasets. We calculate both F1-scores to compare our result with different published results.

Our approach shows very promising results on the PAMAP2 dataset. For PAMAP2, the search took only 10 GPU-hours and the best prediction accuracy was 94.36%. Table 3 compares our algorithm to other published works. In comparison, grid search [15] on CNN for PAMAP2 was able to achieve 93.7% and a hand-crafted CNN [24] was able to reach 93.21% clearly demonstrating that our approach is more efficient than naive search methods such as random or grid search, while also being resource efficient by evaluating far less design points. The best performance was found on a neural network that has 7 convolutional layers followed by 3 fully connected layers (Table 4).

Table 4. PAMAP2 accuracy comparisons

Model	Accuracy (%)	$F1_w$ (%)	$F1_m$ (%)
Hand designed [24]	93.13	93.21	–
Grid search (CNN) [15]	–	–	93.7
D^2C [34]	–	–	92.71
D^2CL [34]	–	–	93.2
EPT (ours)	**94.36**	**94.17**	**94.36**

5 Conclusion

In this paper, we presented a novel approach called Evolutionary Piecemeal Training which traverses the search space of plain CNNs to find an efficient architecture within reasonable constraints for a given task. We validated our algorithm on two different datasets which demonstrates the versatility of our method. We showed that for moderate complexity tasks such as the PAMAP2 dataset, our approach is better and more efficient than random or grid search methodologies.

We perceive that this process can easily be extended to perform multi-objective optimization, which allows the neural network to be optimized simultaneously for resource utilization along with its performance. As future work we aim to modify this search algorithm to incorporate hardware metrics and reduce resource usage of the neural network on the designated embedded system.

Acknowledgements. This project has received partial funding from the European Union's Horizon 2020 Research and Innovation programme under grant agreement No. 780788.

References

1. Jenetics library (2019). https://jenetics.io/
2. Onnx: Open neural network exchange formet (2019). https://onnx.ai/
3. Pytorch: An open source deep learning platform (2019). https://pytorch.org/
4. Baker, B., Gupta, O., Raskar, R., Naik, N.: Accelerating neural architecture search using performance prediction. arXiv preprint arXiv:1705.10823 (2017)
5. Brock, A., Lim, T., Ritchie, J.M., Weston, N.: Smash: one-shot model architecture search through hypernetworks. arXiv preprint arXiv:1708.05344 (2017)
6. Cai, H., Zhu, L., Han, S.: ProxylessNAS: direct neural architecture search on target task and hardware. In: International Conference on Learning Representations (2019)
7. Chen, T., Goodfellow, I., Shlens, J.: Net2net: accelerating learning via knowledge transfer. arXiv preprint arXiv:1511.05641 (2015)
8. Chen, Z., Zhou, Y., Huang, Z.: Auto-creation of effective neural network architecture by evolutionary algorithm and ResNet for image classification. In: 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC), pp. 3895–3900. IEEE (2019)
9. Davies, A., Serjeant, S., Bromley, J.M.: Using convolutional neural networks to identify gravitational lenses in astronomical images. Mon. Not. Roy. Astron. Soc. (2019)
10. Deng, B., Yan, J., Lin, D.: Peephole: predicting network performance before training. arXiv preprint arXiv:1712.03351 (2017)
11. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: a large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition, pp. 248–255. IEEE (2009)
12. DeVries, T., Taylor, G.W.: Improved regularization of convolutional neural networks with cutout. arXiv preprint arXiv:1708.04552 (2017)
13. Elsken, T., Metzen, J.H., Hutter, F.: Efficient multi-objective neural architecture search via Lamarckian evolution. In: International Conference on Learning Representations (2019)
14. Goodfellow, I.J., Warde-Farley, D., Mirza, M., Courville, A., Bengio, Y.: Maxout networks. arXiv preprint arXiv:1302.4389 (2013)
15. Hammerla, N.Y., Halloran, S., Plötz, T.: Deep, convolutional, and recurrent models for human activity recognition using wearables. arXiv preprint arXiv:1604.08880 (2016)
16. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)

17. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4700–4708 (2017)
18. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images. Technical report, Citeseer (2009)
19. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. arXiv preprint arXiv:1608.08710 (2016)
20. Ling, F., et al.: Measuring river wetted width from remotely sensed imagery at the sub-pixel scale with a deep convolutional neural network. Water Resources Research (2019)
21. Liu, H., Simonyan, K., Yang, Y.: Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055 (2018)
22. Lu, Z., et al.: NSGA-Net: a multi-objective genetic algorithm for neural architecture search. arXiv preprint arXiv:1810.03522 (2018)
23. Miikkulainen, R., et al.: Evolving deep neural networks. In: Artificial Intelligence in the Age of Neural Networks and Brain Computing, pp. 293–312. Elsevier (2019)
24. Moya Rueda, F., Grzeszick, R., Fink, G., Feldhorst, S., ten Hompel, M.: Convolutional neural networks for human activity recognition using body-worn sensors. Informatics **5**, 26 (2018)
25. Pan, B., Hsu, K., AghaKouchak, A., Sorooshian, S.: Improving precipitation estimation using convolutional neural network. Water Resour. Res. **55**(3), 2301–2321 (2019)
26. Pham, H., Guan, M., Zoph, B., Le, Q., Dean, J.: Efficient neural architecture search via parameter sharing. In: International Conference on Machine Learning, pp. 4092–4101 (2018)
27. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 4780–4789 (2019)
28. Real, E., et al.: Large-scale evolution of image classifiers. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70, pp. 2902–2911 (2017). JMLR.org
29. Reiss, A., Stricker, D.: Introducing a new benchmarked dataset for activity monitoring. In: 2012 16th International Symposium on Wearable Computers, pp. 108–109. IEEE (2012)
30. Springenberg, J.T., Dosovitskiy, A., Brox, T., Riedmiller, M.: Striving for simplicity: the all convolutional net. arXiv preprint arXiv:1412.6806 (2014)
31. Suganuma, M., Shirakawa, S., Nagao, T.: A genetic programming approach to designing convolutional neural network architectures. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 497–504. ACM (2017)
32. Szegedy, C., et al.: Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–9 (2015)
33. Wang, J., Chen, Y., Hao, S., Peng, X., Hu, L.: Deep learning for sensor-based activity recognition: a survey. Pattern Recogn. Lett. **119**, 3–11 (2019)
34. Xi, R., Hou, M., Fu, M., Qu, H., Liu, D.: Deep dilated convolution on multimodality time series for human activity recognition. In: 2018 International Joint Conference on Neural Networks (IJCNN), pp. 1–8. IEEE (2018)
35. Xie, L., Yuille, A.: Genetic CNN. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1379–1388 (2017)
36. Zhang, H., Cisse, M., Dauphin, Y.N., Lopez-Paz, D.: mixup: beyond empirical risk minimization. arXiv preprint arXiv:1710.09412 (2017)

37. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578 (2016)
38. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 8697–8710 (2018)