Improving Energy Efficiency of Convolutional Neural Networks on Multi-core Architectures through Run-time Reconfiguration

Y. Xiong*, J. Li*, D. Blaauw†, H.-S. Kim†, T. Mudge†, R. Dreslinski†, and C. Chakrabarti*
*School of ECEE, Arizona State University, Tempe
†Dept. of EECS, University of Michigan, Ann Arbor

2022 IEEE International Symposium on Circuits and Systems (ISCAS) | 978-1-6654-8485-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/ISCAS48785.2022.9937275

***Abstract*—Convolutional neural networks (CNNs) are built with convolution layers which account for most of their computation time. The differences in the convolution kernel types (2D, point-wise, depth-wise), and input sizes lead to significant differences in their computation and memory demands. In this work, we exploit run-time reconfiguration to adapt to the differences in the characteristics of different convolution kernels on a low-power reconfigurable architecture, Transmuter. The architecture consists of light-weight cores interconnected by caches and crossbars that support run-time reconfiguration between different cache modes – shared or private, different dataflow modes – systolic or parallel, and different computation mapping schemes. To achieve run-time reconfiguration, we propose a decision-tree-based engine that selects the optimal Transmuter configuration at a low cost. The proposed method is evaluated on commonly-used CNN models such as ResNet18, VGG11, AlexNet and MobileNetV3. Simulation results show that run-time reconfiguration helps improve the energy efficiency of Transmuter in the range of 3.1×-13.7× across all networks.**

***Index Terms*—Energy-efficiency, CNN, runtime reconfiguration, multicore architecture**

I. Introduction

Convolutional Neural Networks (CNNs) have shown superior performance in multiple domains, such as image processing, computer vision, and natural language processing. This has led to the implementation of these models on not only CPUs and GPUs but also low-power end devices. Several application-specific integrated circuits for CNN accelerators that maximize both energy efficiency and performance have been proposed [1]–[4]. Many of these accelerators use systolic arrays [5] for 2D convolution since this is by far the most dominant computation kernel.

The input data size and convolution kernel size vary from network to network and also across layers in a single network. Interestingly enough, while some input-convolution kernel size combinations have the best performance on pipelined systolic arrays, others have the best performance on multicores with the shared cache. Thus it is desirable to have a flexible architecture that can support the configuration that is best suited for the specific layer parameter values.

Field programmable gate arrays (FPGA) are one of the most widely used reconfigurable architectures. They have been shown to achieve promising performance on accelerating DNN algorithms [6]–[8]. Another viable option for neural networks is coarse-grained reconfigurable architectures (CGRA) which achieve high energy efficiency while maintaining programmability [9]–[11]. However, none of these architectures change their computation pattern from layer to layer.

In this work, we use a low-power reconfigurable architecture, Transmuter [12], to demonstrate how performance can be improved by choosing the architectural configuration that best matches the layer-level characteristics of the CNN model. Transmuter consists of clusters of programmable processing units with reconfigurable memory. It can reconfigure the on-chip memory type (shared/private/hybrid), dataflow (parallel/systolic) and mapping scheme (distributed/shared inputs and weights) at run-time with very low overhead ($\approx$ 10 cycles). For 2D convolution, the specific Transmuter configuration with the best performance varies from layer to layer. For example, for VGG11 on CIFAR-10, parallel dataflow with shared cache is the best performing configuration for the first layer while systolic data flow with hybrid cache is the best for intermediate layers where the channel numbers are large. Thus to maximize the performance in every layer for different CNN models, the Transmuter configuration has to be set to the one with the best performance.

Our approach is to first determine the best Transmuter configuration for a selected set of combinations of input and kernel sizes and then derive a decision-tree-based predictor to determine the best configuration at run-time. While the best configuration for all combinations can be stored in a look-up table if the number is small, this method is not scalable. The use of the decision tree helps predict the best configuration even when the layer parameters do not exactly match. Our evaluation shows that the predictor has high accuracy on choosing the right implementation and can achieve 3.12× to 13.31× higher energy efficiency compared to the baseline Transmuter configuration on benchmark CNN models. The key contributions are:

- Developed energy-efficient implementations of 2D convolution on various Transmuter configurations.
- Designed a decision-tree-based predictor for deriving Transmuter configuration at run-time.
- Achieved, on average, 8.97× speedup and 8.7× increase in energy efficiency on benchmark CNN models compared to the baseline.

II. Background

A. 2D Convolution Basics

2D convolution (Conv2D) is the key kernel in convolution neural networks (CNN). It accounts for 93.8% of execution time in ResNet18 and 92.4% in VGG11. A Conv2D computation is shown in Figure 1. The convolution kernel slides through all locations of the input array and carries out multiply and accumulate (MAC) operations between the kernel and input values. The parameters of a Conv2D kernel include the input array size ($N_x \times N_y$), kernel size ($k_x \times k_y$), number of input channels (IC), and number of output channels (OC). In CNN models such as ResNet and VGG 16, the input feature map sizes vary from 224×224 to 7×7, and the input and output channel numbers vary from 16 to 2048. The most popular kernel sizes are 3×3 and 1×1.

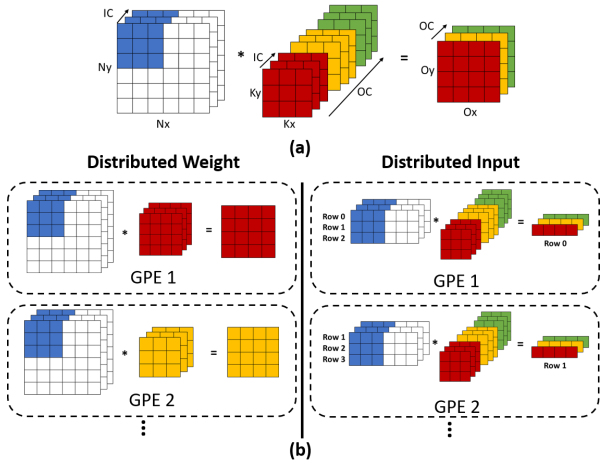

Fig. 1. a) Conv2D Computation. b) Distributed Weight and Distributed Input Modes for GPE-level Computations

B. Transmuter

Transmuter is a low-power programmable architecture that is reconfigurable [12]. It consists of multiple tiles of in-order general-purpose processing (GPE) units, distributed on-chip cache memories, crossbars and a high-bandwidth DDR interface, as is shown in Figure 2. Each GPE is a small processor with floating-point (FP) and load/store (LS) units and uses a standard Arm ISA. Each tile consists of several GPEs and one local control processor (LCP) which is primarily responsible for distributing work across the GPEs. The GPEs can be configured to operate in parallel mode or systolic array mode. The L1 and L2 data caches can be configured in shared/private cache mode and hybrid cache-scratchpad memory (SPM) mode, where half of the cache banks are configured as shared cache and the other half operate as SPM. Transmuter can reconfigure the on-chip memory type, resource sharing, and dataflow at run-time with a latency of 10 cycles.

III. Method Description

A. Run-time Reconfiguration parameters

The optimal architecture configuration depends on the layer parameter values. For example, for layers with an input size

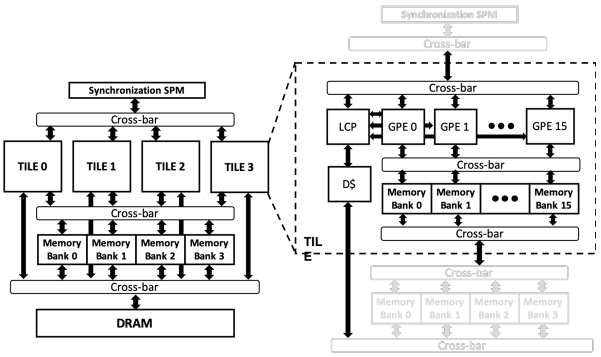

Fig. 2. Transmuter Architecture Block Diagram

of $224 \times 224 \times 3$ (the input size of ImageNet dataset), shared cache mode with parallel dataflow is the optimal configuration that has more than $5\times$ lower latency compared to systolic dataflow with hybrid cache mode implementations. But when it comes to intermediate layers with smaller input sizes and larger input channel numbers, such as the last residual block in ResNet18, the systolic dataflow with hybrid mode implementation can significantly accelerate the computation time by more than $10\times$. Thus, by choosing the hardware configuration and dataflow mapping, the implementation of all conv2D kernels can be optimized. In this work, the following reconfigurable options are considered:

Cache configuration: Use shared cache, private cache, and hybrid cache (combination of cache and scratchpad memory) for both L1 and L2. The shared cache mode increases the effective cache size for L1/L2 hierarchy while the private cache mode avoids multi-core cache access conflict. The hybrid cache mode assigns half of the cache as scratchpad memory (SPM), which is software-managed, and the other half as shared cache. In cases where the frequently reused values are known apriori, SPM increases the data access efficiency.

Dataflow: Parallel or register-to-register (R2R) systolic. In parallel dataflow, each GPE works independently. It loads the input and weight values, computes, and writes back the output values. In R2R systolic dataflow, each GPE in a systolic array computes a local partial sum and passes the partial sum to the next GPE in the array via the R2R data path. The last GPE in the array computes and writes back the final output value to the main memory.

Tile-level mapping: Distribute the computation across multiple Transmuter tiles using either distributing weight mode or distributing input mode. In the distributed weight mode, the input array is shared across all tiles, the weights are distributed to each tile and each tile is responsible for computing the output channels that correspond to its assigned set of weights. In the distributed input mode, the input array is partitioned by rows and assigned to each tile. The weights are shared across tiles and each tile is responsible for the outputs that correspond to its assigned set of input rows. For 3×3 convolution kernel size, when IC is small, distributed weight mode performs better, and when IC becomes larger, distributed input mode

has better performance.

GPE-level computations: Here the computations assigned to a tile are distributed across multiple GPEs or multiple groups of GPEs (where each group corresponds to a systolic array) in a tile. The two modes are shown in Figure1 (b). In the distributed weight mode, the weights assigned to the tile are distributed to the GPEs in that tile; each GPE computes the output channels that correspond to the assigned weights. In the distributed input mode, each GPE in the tile computes the output rows using shared weights and the input rows assigned to it. The choice of the computation scheme is highly dependent on the sizes of input and weight assigned to each tile. When the size of the input is large, distributed input mode performs better, and when the size of weight is large, distributed weight mode has better performance.

B. Decision Tree Engine

CNN models have a large diversity in the sizes of the inputs and weights – they differ from network to network and across layers in a network. Typically, the initial layers tend to have larger input sizes and the later layers tend to have larger weight sizes. We found that the shape of the input and weight affects the performance of different Transmuter configurations.

To determine the optimal Transmuter configuration at run time, we derive a predictor that provides a mapping from the input parameter space to the configuration space. Since determining the Transmuter performance for every possible combination of input size and weight size for all candidate networks is not possible, we only consider a select subset of the input parameter space where the input sizes are powers of 2, and the convolution kernel sizes are 3×3 and 1×1.

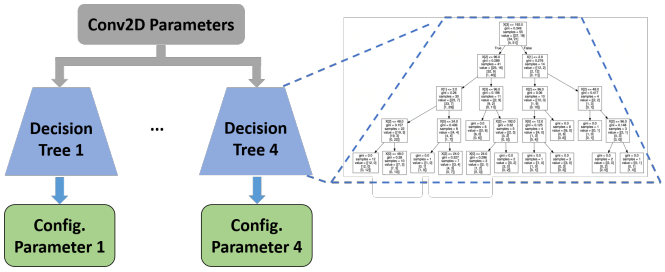

Fig. 3. Predicting Transmuter configuration parameters using a decision tree-based predictor

We propose a predictor using the decision tree method to derive the optimal mapping of Conv2D layers. Decision tree based predictors are lightweight and have been shown to have good prediction performance. A recent example is SparseAdapt [13] which was used to predict the optimal Transmuter configuration for sparse matrix-matrix and matrix-vector multiplication. We evaluated predictors built with one multi-output decision tree and multiple single-output decision trees and found that multiple single-output decision trees for each configuration parameter lead to better performance. Figure 3 shows part of the decision tree model that takes the parameters of Conv2D layers as input and generates the optimal configuration parameters (cache configuration, dataflow, tile-level mapping and GPE-level computation). The model is executed on the host machine before the CNN computation starts.

IV. Evaluation and Performance

A. Platform configuration

In this work, we use a Transmuter architecture with 4 tiles, each consisting of one LCP and 16 GPEs. The LCP and GPEs operate at 1.0 GHz. The L1 cache consists of 16 4-kB cache banks and the L2 cache consists of 4 4-kB cache banks. The simulations are carried out using gem5 simulator [14] and the simulation results include execution time and floating-point operation numbers, etc.

The power consumption is computed as the sum of static power and dynamic power of all Transmuter components. The power consumed by the processing unit is modeled using the Arm Cortex M4F core specification. The static power and transaction energy per access of the reconfigurable cache banks are modeled using CACTI 7.0 [15]. The power parameters are optimized by comparing to the power measurements of a taped-out 4x8 Transmuter system in 28nm [16] and scaled to 14nm node.

B. Decision Tree Engine Validation

To develop the decision tree engine, we first collected a dataset of conv2D layers parameters and their corresponding Transmuter configuration that achieved the lowest latency and highest energy efficiency. Each sample is an array that consists of input feature map size I, kernel size F, number of input channels IC, and number of output channels OC. The optimal configuration is represented by a 4-dimensional array with the following encodings: For cache configuration, shared L1 cache is 0 and hybrid L1 cache is 1; for dataflow, the parallel mode is 0 and R2R systolic mode is 1; for both tile-level mapping and GPE-level computation, distributed input is 0 and distributed weight is 1. We represented kernel sizes of 3×3 and 1×1 by $F = 3$ and $F = 1$, respectively. We used the power of 2 numbers for IC, OC, and I, namely, 8 to 256 for IC, 16 to 512 for OC, and $I = 8$ to $I = 128$ to represent input feature maps of sizes 8×8 to 128×128, respectively. The total number of samples is 230.

We used the Scikit-Learn python library [17] to train the four decision trees. We evaluated the performance of decision tree models with different depths and found that the prediction accuracy increases as the depth increases. Since the accuracy performance does not improve for depth larger than 11, we fixed the depth to be 11.

To validate the robustness of the model, we randomly split the dataset into a training set (200 samples) and a testing set (30 samples) in each evaluation. We consider the model to be correct when the predicted configuration matches the optimal configuration completely. The reported accuracy is the average accuracy from 100 evaluations.

The proposed predictor achieves 83.26% prediction accuracy. Figure 4 lists the samples where the predicted and optimal configurations do not match. We see that the energy

efficiency due to incorrect predictions is not too different from that of the optimal configuration. The difference varies from 1.2%-7.1% with an average of 5.2%.

I	F	IC	OC	Predicted	Optimal	Predicted GFLOPS/W	Optimal GFLOPS/W	Diff.
128	3	32	64	[0001]	[0000]	13.13	13.97	6.0%
32	3	32	128	[1111]	[1110]	17.97	19.35	7.1%
8	3	128	256	[1111]	[1101]	17.29	17.50	1.2%
32	1	16	128	[1011]	[1101]	20.18	21.26	5.1%
8	1	256	256	[0011]	[0001]	16.34	17.51	6.7%

Fig. 4. Incorrectly predicted samples only have a small difference in GFLOPS/W, with average difference of 5.2%

C. Network Evaluation

Next we evaluate the benefit of decision engine-aided run-time reconfiguration on several commonly used CNN benchmarks, namely AlexNet [18], VGG11 [19], MobileNetV3 [20] and ResNet18 [21]. We use energy efficiency (GFLOPS/W) and latency as the evaluation metrics. The *baseline* Transmuter configuration is a shared cache-based parallel architecture with the GPEs computing using distributed weights and shared inputs; it does not support runtime reconfiguration. Table I lists the speedup, energy, and energy efficiency (GFLOPS/W) obtained by using run-time reconfiguration compared to the baseline for the four CNN models.

TABLE I
NETWORK EVALUATION RESULTS

Model	Speed Up	Energy↓	Energy Efficiency↑
AlexNet	9.26×	9.26×	8.95×
VGG11	13.60×	12.00×	13.71×
MobileNetV3	3.32×	3.25×	3.12×
ResNet18	9.72×	9.06×	9.06×

AlexNet is a CNN model that uses 5 conv2D layers with I varying from 13 to 224, IC varying from 3 to 384, and OC varying from 64 to 384. The second layer with $I = 56, IC = 64, F = 5, OC = 192$ is the most time-consuming layer and accounts for 58.62% of the overall execution time. The hybrid mode with R2R systolic dataflow decreases the execution time of this layer from 2.28 s to 0.28 s. Layer-level reconfiguration speeds up the execution by 9.26× and increases the energy efficiency by 8.95×.

VGG models proposed for the CIFAR-10 data set have an input image size of $32 \times 32 \times 3$. The intermediate layers (4 to 6) have large channel numbers, ranging from 64 to 512, which negatively affect the L1 cache performance of the baseline configuration. The predicted configuration uses hybrid cache mode with R2R systolic dataflow; the GPEs in the array compute using distributed weights and shared inputs. Overall, layer-level reconfiguration helps achieve 13.60× speedup and 13.71× increase in energy efficiency.

MobileNetV3 models simplify the Conv2D computation using separable 2D convolution, where a convolution layer is replaced with a depth-wise convolution and a point-wise (1×1) convolution. The baseline configuration spends 66 ms on point-wise conv2D layers, which is 90.6% of the overall execution time. In comparison, the predicted configuration uses hybrid cache mode with parallel or R2R systolic dataflow (depending on the layer) and decreases the execution time to 16 ms. With run-time reconfiguration, MobileNetV3 achieves 3.32× speedup and increases energy efficiency by 3.12×.

ResNet18 is an 18-layer CNN that uses residual bypass to prevent the gradient from vanishing. The predicted configuration implements the 3×3 Conv2D layers using the shared cache with parallel dataflow in the first layer and hybrid mode with systolic dataflow for all other layers. For 1×1 Conv2D, hybrid mode with parallel dataflow is used. Overall, run-time reconfiguration speeds up ResNet18 by 9.72× and increases the energy efficiency by 9.06×.

Config.	AlexNet	VGG 11	MobileNetV3	ResNet 18
0-0-1-1	5.90%	1.36%	14.49%	-
0-0-0-1	-	-	-	5.76%
1-1-0-1	65.15%	34.45%	29.47%	39.81%
1-1-1-1	25.37%	45.01%	2.72%	-
1-1-1-0	-	-	-	18.24%
1-0-1-1	-	11.58%	13.51%	30.00%
1-0-0-1	-	-	27.01%	-

Fig. 5. Percentage of time spent in each of the configurations

Figure 5 shows the percentage of time spent in each Transmuter configuration. The results show that the four CNN models use only 3 to 5 distinct configurations that differ in cache configuration, dataflow, and data mapping. The most popular configuration is the systolic array with hybrid cache mode (1-1-x-x). It accounts for 90.52% for AlexNet, 79.46% for VGG11, 32.19% and 58.05% for ResNet18.

V. CONCLUSION

This paper investigates using run-time reconfiguration to improve the energy efficiency of CNN implementations on a low-power reconfigurable architecture, Transmuter. CNN models use conv2D layers of different sizes and the optimal Transmuter implementation changes as the size changes. To determine the optimal implementation for Transmuter at runtime, we proposed a decision-tree-based configuration predictor. Our evaluation shows that the predictor achieves an average prediction accuracy of 83.26% and the predicted configurations have little performance difference compared to the optimal configuration. We evaluate the performance of the predictor on 4 popular CNN models. Run-time reconfiguration with configuration predictor helps achieve a speedup of 3.32× to 13.60× and energy efficiency increase of 3.12× to 13.71× compared to the baseline model.

ACKNOWLEDGMENT

This paper was supported in part by a grant from AFRL and DARPA under agreement number FA8650-18-2-7864.

REFERENCES

[1] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *ACM Sigplan Notices*, vol. 49, no. 4. ACM, 2014, pp. 269–284.

[2] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun *et al.*, "Dadiannao: A machine-learning supercomputer," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2014, pp. 609–622.

[3] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," in *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3. ACM, 2015, pp. 92–104.

[4] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[5] H.-T. Kung, "Why systolic architectures?" *Computer*, vol. 15, no. 01, pp. 37–46, 1982.

[6] X. Wei, Y. Liang, and J. Cong, "Overcoming data transfer bottlenecks in FPGA-based DNN accelerators via layer conscious memory management," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019, pp. 1–6.

[7] C. Hao, X. Zhang, Y. Li, S. Huang, J. Xiong, K. Rupnow, W.-m. Hwu, and D. Chen, "FPGA/DNN co-design: An efficient design methodology for IoT intelligence on the edge," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019, pp. 1–6.

[8] Y. Chen, J. He, X. Zhang, C. Hao, and D. Chen, "Cloud-DNN: An open framework for mapping DNN models to cloud FPGAs," in *Proceedings of the 2019 ACM/SIGDA international symposium on field-programmable gate arrays*, 2019, pp. 73–82.

[9] Z. Yuan, Y. Liu, J. Yue, J. Li, and H. Yang, "Coral: coarse-grained reconfigurable architecture for convolutional neural networks," in *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2017, pp. 1–6.

[10] D. Liu, S. Yin, G. Luo, J. Shang, L. Liu, S. Wei, Y. Feng, and S. Zhou, "Data-flow graph mapping optimization for cgra with deep reinforcement learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 12, pp. 2271–2283, 2018.

[11] Y. Li and A. Pedram, "Caterpillar: Coarse grain reconfigurable architecture for accelerating the training of deep neural networks," in *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2017, pp. 1–10.

[12] S. Pal, S. Feng, D.-h. Park, S. Kim, A. Amarnath, C.-S. Yang, X. He, J. Beaumont, K. May, Y. Xiong *et al.*, "Transmuter: Bridging the efficiency gap using memory and dataflow reconfiguration," in *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, 2020, pp. 175–190.

[13] S. Pal, A. Amarnath, S. Feng, M. O'Boyle, R. Dreslinski, and C. Dubach, "Sparseadapt: Runtime control for sparse linear algebra on a reconfigurable accelerator," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 1005–1021.

[14] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[15] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 2, p. 14, 2017.

[16] S. Kim, M. Fayazi, A. Daftardar, K.-Y. Chen, J. Tan, S. Pal, T. Ajayi, Y. Xiong, T. Mudge, C. Chakrabarti *et al.*, "Versa: A dataflow-centric multiprocessor with 36 systolic arm cortex-m4f cores and a reconfigurable crossbar-memory hierarchy in 28nm," in *2021 Symposium on VLSI Circuits*. IEEE, 2021, pp. 1–2.

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[19] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *International Conference on Learning Representations*, 2015.

[20] 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.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.