Proceedings Paper:
Wang, Ziwei, Trefzer, Martin A orcid.org/0000-0002-6196-6832, Bale, Simon J et al. (1 more author) (2022) A Multi-objective Evolutionary Approach for Efficient Kernel Size and Shape for CNN. In: 2022 International Joint Conference on Neural Networks (IJCNN). IEEE.

https://doi.org/10.1109/IJCNN55064.2022.9892201

A Multi-objective Evolutionary Approach for Efficient Kernel Size and Shape for CNN

Ziwei Wang, Martin A. Trefzer, Simon J. Bale, Andy M. Tyrrell
Department of Electronic Engineering
University of York
York, United Kingdom
zw960, martin.trefzer, simon.bale, andy.tyrrell@york.ac.uk

***Abstract*—In order to improve the classification accuracy, state-of-the-art CNNs usually involve large and complex convolutional layers. However, for certain applications, e.g. Internet of Things (IoT), where such CNNs are to be implemented on resource-constrained platforms, the CNN architectures have to be small and efficient. To deal with this problem, reducing the resource consumption in convolutional layers has become one of the most significant efforts when designing CNNs. In this work, a multi-objective optimisation approach is proposed to trade-off between the amount of computation and network accuracy by using Multi-Objective Evolutionary Algorithms (MOEAs). The size and number of convolution kernels used are proportional to computational resource consumption of CNNs. Therefore, this paper considers reducing the computational resource consumption by reducing the size of kernels in convolutional layers. In addition to kernel size unconventional kernel shapes are investigated and results show these clearly improve performance over using traditionally square convolution kernels alone. The main contributions of this paper are therefore a methodology to significantly reduce computational cost of CNNs, based on unconventional kernel shapes, and provide different trade-offs for specific use cases. The experimental results further demonstrate that the proposed method achieves large improvements in resource consumption with no significant reduction in network performance. Compared with the benchmark CNN, the best trade-off architecture shows a reduction in multiplications of up to 4.06X and with slight increase in classification accuracy on CIFAR-10 dataset.**

***Index Terms*—Convolutional Neural Networks, Evolutionary Computation, Deep Learning, Multi-Objective Evolutionary Optimisation**

I. Introduction

Deep Neural Networks (DNNs) are biologically-inspired computing systems which consist of internal connections between multiple layers of artificial neurons. In recent years, DNNs have drawn increasing attention in various application domains, such as image processing, speech recognition and many other challenging Artificial Intelligence (AI) tasks [1]. In particular, Convolutional Neural Networks (CNNs) have successfully gained outstanding performance in image classification and video recognition [2]–[4]. In general, the common CNN architecture usually consists of convolutional layers and pooling layers in between its input layer and fully-connected output layer. Compared with the conventional fully-connected neural network, neurons in a convolutional layer are only connected with several neurons in the previous layer. Specifically, any neuron in the feature map of a convolutional layer is a linear combination of neurons in the receptive field which is defined by a convolution kernel in the previous layer [5], [6]. The state-of-the-art achievements of designing CNN architectures have demonstrated significant improvements on network classification accuracy on various benchmark datasets, such as GoogleNet [7] and AlexNet [8]. In order to get the best accuracy on specific tasks, the computational complexity of the network grows exponentially and this brings new challenges of executing it on resource-constrained platforms. Based on the analysis by [9], several computationally low-cost network architectures can still achieve reasonable accuracy on state-of-the-art benchmark datasets. The motivation to find CNN architectures that can achieve high accuracy and keep computational resource usage at a minimum is therefore high.

Multi-objective evolutionary algorithms (MOEAs) are today one of the most commonly used methodologies in solving hard optimisation problems where two or more (multiple) conflicting objectives must be satisfied simultaneously [10]. MOEAs aim to generate a set of possible solutions (so-called population) in a single run of the algorithm. In recent years, there have been several research studies focusing on optimising neural network typologies and their connection weights. These approaches have demonstrated that using evolutionary algorithms to evaluate neural network topology can achieve competitive performance on state-of-the-art benchmark datasets [11]–[13].

In this article, we apply MOEAs to the optimisation of feed-forward CNN architectures for use in resource-constrained scenarios. In order to achieve a high classification accuracy, state-of-the-art CNN architectures are extremely complex. For example, AlexNet [8] requires billions of multiply-accumulation (MAC) operations to process a single image. In general, convolutional layers are the most computationally expensive ones, where the resource consumption in each layer is proportional to the number and sizes of the convolution kernels. In order to minimise the resource consumption and retain the network accuracy while processing the CNNs on the target hardware platform, we use the fast Non-dominated Sorting Genetic Algorithm (NSGA-II) [14] to explore the design space for the feed-forward CNN architecture and produce the best trade-off between network complexity and

classification accuracy. Our proposed method is focusing on optimising convolutional layers using combinations of various unconventional shapes and sizes of kernels, while keeping the network depth constant. The proposed method is tested on widely-used benchmarks, MNIST [15], Fashion-MNIST [16] and CIFAR-10 [17], and compared with conventional CNNs.

The paper is organised as follows: Section II gives an overview of the current approaches of optimising feed-forward neural network architecture, as well as optimising learning algorithms and hyperparameters. Section III describes the design methodology that implements NSGA-II to explore the design space of CNN architecture and explains its features. Section IV shows several test experiments with varying benchmark datasets and provides a proof-of-concept of the performance of the proposed method. Finally, Section V concludes the paper and discusses further works.

II. Related Work

This section introduces a review of the current approaches of automatic optimisation techniques that are used to find good solutions by network topology optimisation, such as connection weights, network structure.

A. Optimisation of Network Architecture

Evolutionary Algorithms (EAs) are widely used in optimisation problems with complex fitness landscapes. The main idea of optimising artificial neural networks using EAs is to evolve the synaptic weights and connections of the network [13]. NEAT [18] is a method that uses genetic algorithms (GAs) to change both connection weights and network structure. Their proposed method encodes each neuron and synaptic weight in the genotype. For each iteration, the GA can either add additional neurons to the network or adjust the input/output connections of a neuron. This method allows the GA to find out the best network topology for the target task. A hypercube-based NeuroEvolution of Augmenting Typologies (HyperNEAT) method has shown advantages when optimising weights for CNNs [19], [20]. However, due to the lager search space of CNN topologies, this method requires huge amounts of computational resources. Therefore, this method is difficult to scale up to state-of-the-art deep neural network architectures, because of their size. G. Morse and K. Stanley [21] compared evolutionary algorithms with the stochastic gradient descent (SGD) method for weight optimisation of ANNs. Their results demonstrate that using an evolutionary algorithm to optimise weights achieves competitive results, compared with the traditional SGD method.

It is therefore reasonable to consider GAs to be a suitable method for optimising network topology. However, for network weight optimisation, GAs show almost the same performance as SGD through back-propagation. Therefore, in order to search for efficient neural network architectures and updating network weights at the same time, a combination of GAs and SGD methods have been investigated in recent years. Real et al. [22] apply GAs to CNN design, where the model is trained by SGD through back-propagation and the architecture is optimised by simple GA. They initialise the starting point as a small model which only consists of a single pooling layer. With each evolutionary step, the model is augmented by adding more convolution layers. Their approach is only focusing on network accuracy, therefore, the result shows that as the network accuracy is increased, the computational effort required is increased dramatically. CoDeepNEAT [23] is a further extension of NEAT [18] where the population is separated into two sub-sets: module and blueprint. The module chromosome is a graph that represents a small ANN and the blueprint chromosome is a graph where each mode contains a pointer to a particular module species. During the evolution, the two sub-sets are combined together to build a larger network, where each mode in the blueprint is replaced with a module chosen randomly from the species to which that mode points. Their results show that the network designed by CoDeepNEAT can achieve competitive accuracy in image classification problems with faster training speed. Kim et al. [24] use a multi-objective evolutionary algorithm (MOEA) to trade-off between classification accuracy and run-time. They adopt NSGA-II to explore the Pareto front of the design space. The network architecture is decoded into two categories: number of outputs in each layer and the total number of convolution layers. Their experimental results show that multi-objective optimisation can further reduce the run-time and achieve better accuracy compared with human-expert design. L. Xie and A. Yuille [11] present a GA solution for searching large-scale CNNs. In their work, the GA is applied to designing the network structure, where the connectivity of each layer is encoded by a binary string representation. This method can be easily modified for different network architectures and includes different types of layers and connectivity.

Apart from using GAs to optimise the CNN architecture by pre-processing the initial populations, such as pre-defining the layer functionalities and connectivity, there are also some GA-based fully automatic architecture design methods addressed in recent years, e.g. Sun et al. [25]. Their design methodology contains a building block that directly using a skip layer to replace the convolutional layer. The skip layer contains two convolutional layers and one skip connection, where the skip connection connects the input of the first convolutional layer to the output of the second convolutional layer. Then, a GA is applied to searching suitable connections of skip layers and pooling layers. Finally, fully-connected layers are added to the tail of the CNN. Similarly, Suganuma et al. [12] proposes using Cartesian Genetic Programming (CGP) [26] to represent deep neural network architectures and to use highly functional modules as the node functions to reduce the search space. There are six different node functions in their design, including convolutional blocks, residual blocks, max and average pooling, etc. The CGP encodes CNNs as directed acyclic graphs with a two-dimensional grid of nodes. Their results demonstrate that the architectures built by CGP outperform most of the hand-designed modules and provide a good trade-off between classification accuracy and the number of parameters.

B. Unconventional Convolutions

Traditional CNN designs use square kernels to detect image features. This design method brings significant challenges for computational systems, because the number of arithmetic operations increases exponentially as the network size increases. In order to reduce computational resource usage and speed up large CNNs, recent research using unconventional kernel shapes has focused on approximating existing square-kernel convolutional layers for network compression and acceleration. Recent approaches [27]–[29] have demonstrated that some of the 2-D square convolution kernels can be factorised into two 1-D kernels. For example, if a convolutional layer contains a set of $n \times n$ 2-D kernels, where n represents the kernel height and width, it can be factorised as a sequence of two layers with $n \times 1$ and $1 \times n$ kernels, which uses fewer computations. Therefore, the 1-D convolution approximation can significantly accelerate the classification speed as well as reducing the number of network parameters. Similarly, Jin et al. [30] introduce this into the training phase by factorising a conventional 3-D convolution kernel into three consecutive 1-D kernels. Their results show that by factorising the 3-D convolution layers, the network can be accelerated by approximately a factor of two while sustaining similar or better classification accuracy than using conventional 3-D convolution kernels.

Another approach to design the convolutional layer is to use multiple sizes of kernels in one layer. GoogleNet [7] is one of the most accurate CNNs on state-of-the-art benchmark datasets. In order to increase the network accuracy, their work introduces an inception module to increase the network depth and width. Firstly, the inception module contains multiple differently-sized convolution kernels, as well as max pooling operations. Different types of kernels are computing in parallel and extract features at multiple scales. After that, feature maps from different kernels are concatenated to form the input of the next module. In order to reduce the computational effort, a 1×1 kernel convolution is inserted into the inception module for dimension reduction. Hence, the network can grow deeper and wider with reasonable increase in computational resource usage. Szegedy et al. [31] modify the inception module, where the 7×7 convolution kernels are replaced by a sequence of 7×1 and 1×7 kernels, so that the overall computing resources are reduced when compared with the original design. Ding et al. [32] propose Asymmetric Convolution Blocks (ACBs) to replace the conventional convolution kernels. Typically, ACBs replace the conventional convolutional layer with three parallel layers, where these layers contain $n \times n$, $n \times 1$ and $1 \times n$ kernels respectively. Finally, the outputs from each layer are summed up to enrich the feature space.

III. Methodology

This work considers how the classification accuracy of CNNs can be improved (or kept the same) while significantly reducing the computational resources required. Literature of previous investigations suggests that using different shapes of convolution kernels can improve the network performance by detecting multi-scale features from input images [32]. In this section, an analysis into how the size and shape of the convolution kernels are processed regarding to their computational resource consumption is conducted. Then, a set of different shapes of convolution kernels is designed to be used in convolutional layers. The MOEA used is NSGA-II [14] to automatically discover efficient network architecture by finding the trade-off between the computational complexity and module classification accuracy on the three benchmark datasets.

A. Computational Resource Consumption

For state-of-the-art CNN architectures, the computationally most expensive parts are the convolutional layers [9]. In a feed-forward CNN, the convolutional layers are used to extract features from input images. The conventional 2-D convolution process is illustrated in Fig. 1. Mathematically, the formulae for calculating a single feature map by convolution operation can be described as (1),

$$O_{:,:,oc} = \sum_{i=1}^{IC} I_{:,:,i} \otimes K_{:,:,oc} \tag{1}$$

where $O_{:,:,oc}$ is an output feature map of the convolutional layer in the oc-th channel, I is the input image of the convolutional layer with IC channels, K is the set of 2-D convolution kernels in current layer and $\otimes$ represents the convolution operation. It can be seen from the equation, the calculation consists of repeatedly accumulating multiple kernels to form a number of feature maps, which represent the different characteristics of the input image.

Processing a CNN in hardware requires multiply-accumulation (MAC) operations to obtain the output feature maps. In order to improve the classification accuracy of CNNs, state-of-the-art CNN architectures have become increasingly complex, therefore, it is necessary for CNN designs to consider their computational resource consumption.

For a convolutional layer, the number of MAC processes is contingent on the size of the convolution kernels, number of kernels and the size of the output feature maps. It can be calculated using following equation:

$$Operation_{conv} = O_h \times O_w \times O_c \times K_h \times K_w \times K_c \tag{2}$$

where O_h and O_w indicate the height and width of the output feature maps and O_c represents the number of output feature maps, i.e. output channels. Similarly, K_h, K_w and K_c indicate the height, width and the number of the convolution kernels in the corresponding layer. In the conventional design methodology of CNNs, feature maps and convolution kernels are always square, which means the height and width of the convolution kernels and feature map are the same.

In a CNN, the fully-connected layer takes the output of the previous convolution processes and predicts the best classification that is labelled to describe the image. Equivalent to (2), the number of operations in the fully-connected layer of a specific CNN can be calculated as:

$$Operation_{fc} = O_h \times O_w \times O_c \times N_i \tag{3}$$

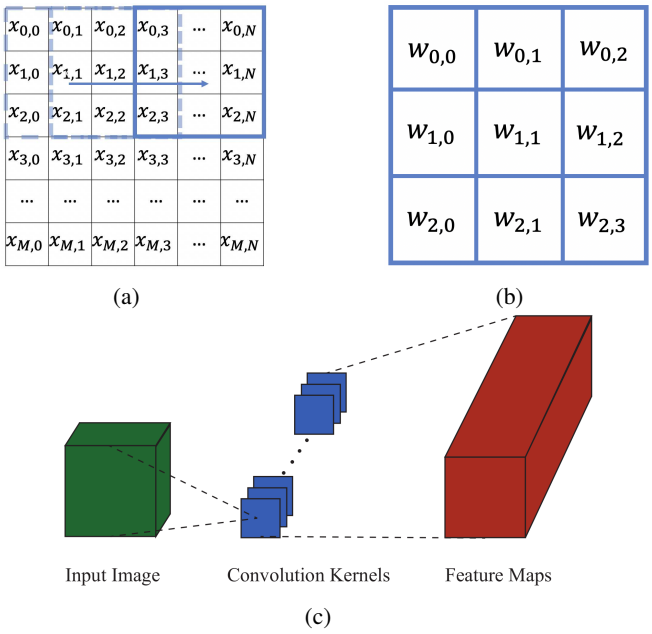

Fig. 1: (a) An example of sliding a 3×3 convolutional kernel across the input image. (b) A conventional 3×3 square kernel that is used to extract features from the input image. (c) Example of a conventional convolutional layer with multiple square kernels.

where the O_h, O_w and O_c indicate output feature maps from the last convolutional or pooling layer and N_i is the number of neurons in the fully-connected layer.

As can be seen from (2) and (3), the convolutional layer requires more MAC operations than the fully-connected layer. Therefore, reducing the number of MAC operations can significantly reduce the computational resource consumption overall. This is important to allow state-of-the-art CNNs to be processed on resource-constrained platforms, such as FPGAs and embedded devices. Apart from the convolutional layers and fully-connected layers, the CNN architecture also involves other layers, including average pooling, max pooling or batch normalisation. For instance, average pooling is used to calculate the average number of pixels in the kernel and max pooling is proposed to find the maximum number of input pixels, which are smaller than operations for convolutional layers in real cases.

B. Mixed Unconventional Kernels

In this work, in order to minimise the hardware costs while maintaining the CNN classification accuracy, the MAC operations in the convolutional layers are minimised by reducing the kernel sizes, i.e. the product of K_h and K_w in (2). For example, a 2-D 3×3 convolution kernel can be replaced by a 3×1 kernel followed by 1×3 kernel, which reduces the number of operations from 9 to 6. However, previous research suggests that the replacement is not equivalent as it does not work as well on some of the lower level layers [31], and not all possible 3×1 kernels are captured by the decomposition. Hence, such a substitution requires the network to have extra kernels or layers to compensate, which may potentially increase again the computational complexity. Therefore, the first question to investigate here is what kinds of kernels can be replaced by smaller ones in a convolutional layer. In addition, without adding an additional convolution kernels, it is investigated whether the conventional 2-D square kernels can be directly replaced by more generic sizes of $m \times n$ shape kernels. Finally, the best combination of different shapes and sizes of kernels for each convolutional layer is considered.

Utilising different kernel sizes allows the network to extract features at multiple scales [7]. It is becoming more popular for state-of-the-art CNN architectures to use small kernels, such as 3×3 and 5×5. Therefore, the largest kernel selected for the network architecture proposed here has a size of 5×5. In order to find the best combination of kernel shapes in a convolutional layer, conventional square kernels and 1-D kernels are used as well as other sizes of kernels, such as 5×3 and 1×1. In total there are 9 different sizes of kernels considered here, the largest being 5×5 and the smallest one 1×1, as shown in Fig. 2.

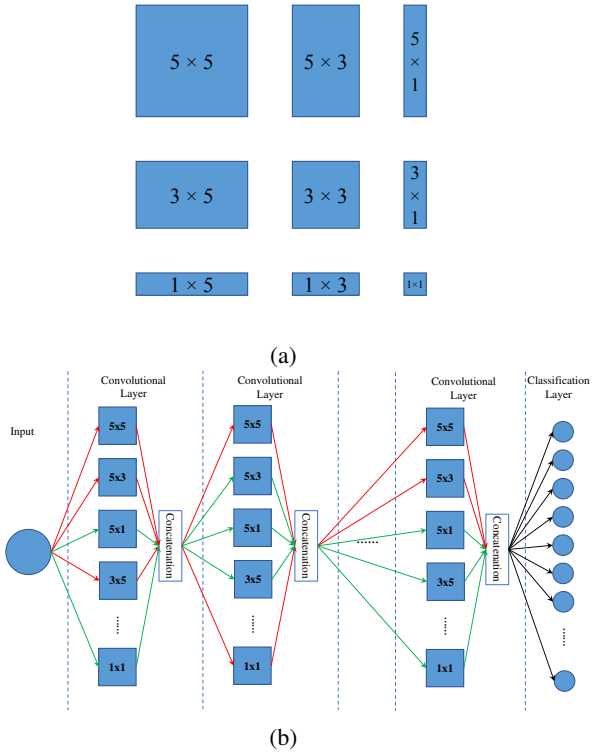

Fig. 2: (a) The set of unconventional kernels. The number of operations required to compute each of the unconventional kernels can be calculated by kernel height $\times$ width. (b) Format of the genotype: The Red and Green Lines shows two different individuals which represent different connections between different size and shape of kernels.

Previous approaches have shown that it is possible to use a series of kernels with differing sizes to better handle multiple-scale objects in a convolutional layer [7]. Rather than using

conventional square kernels, this work puts the set of 1-D stripe kernels into a single convolutional layer. This is so that the network can operate in parallel on different sizes with the most accurate detailing, 1×1, to the biggest kernels, 5×5. Then, all feature maps generated from the different kernels are concatenated together into a single output tensor in order to form the input of the next stage. Padding strategies are applied to make sure that the output feature maps from each set of unconventional kernels will have the same resolution as others. Then, the concatenation are used to combine the output feature maps from each set of unconventional kernels into one output tensor. The overall architecture of the proposed design can be viewed as Fig. 3.

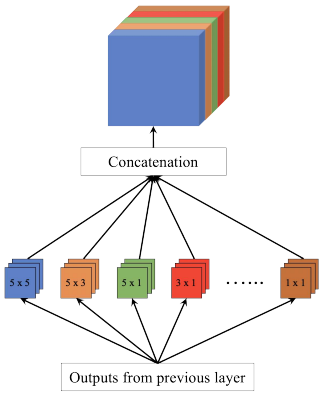

Fig. 3: Overall design of the convolutional layer which involves multiple sizes of kernels to produce different feature maps. If the strides of each type of kernel are the same, using the padding method, the output from each set of kernels will be the same as other kernels. Therefore, the final output from this layer can be concatenated into a single tensor with no additional computation operation required.

C. Multi-Objective Evolutionary Optimisation

Following the convolutional layer design methodology from the previous section, it is difficult to define the optimal kernel sizes required to replace the conventional square kernels in a given CNN architecture. The new methods described here propose to use the non-dominated sorting genetic algorithm (NSGA-II) to explore the kernel size design space of the CNN architecture. NSGA-II is one of the most popular multi-objective optimisation algorithms which uses a fast non-dominated sorting approach and diversity preservation [14]. The approach of optimising for Pareto optimality makes it possible to trade-off between network accuracy and hardware resource consumption. An overview of optimising a given CNN architecture is shown in Fig. 4.

In the optimisation loop, each of the possible unconventional kernel shapes is identified with its kernel ID that represents a specific kernel. As shown in Fig. 4, the optimisation loop takes a given conventional CNN architecture as its input.

The kernels from all layers of the input CNN are encoded in the genotype. As we are focusing on optimising the

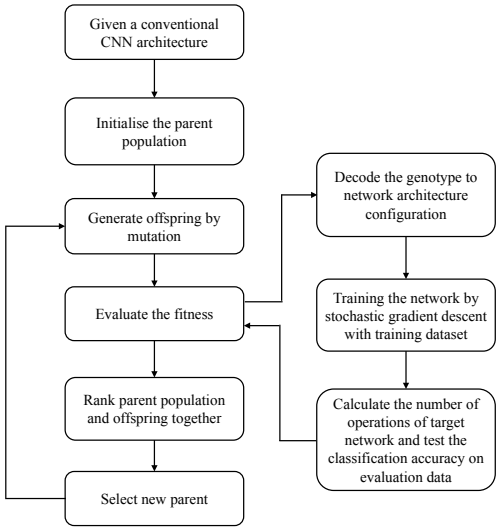

Fig. 4: Overview of the multi-objective optimisation loop. The method selects a set of unconventional kernel shapes replacing the original (conventionally used) square kernels in a given CNN architecture. Each individual is trained on a training data set. Then, the fitness is calculated and assigned based on the classification accuracy on the evaluation data set (objective 1) and the number of arithmetic operations (objective 2). NSGA-II searches for the Pareto front that trades-off between of number operations and model classification accuracy.

convolutional layers, other hyperparameters of the network are kept the same as in the input CNN, e.g. other types of layers, stride, activation function and number of layers. The initial parent population of NSGA-II is generated by randomly replacing the original square kernels with randomly selected unconventional kernel sizes for all convolutional layers in the network. After the initial parent population is created, the first offspring population is generated by mutation operation, which changes the shape of randomly selected kernels based on a given probability. In this case, the genetic representation only consists of a single chromosome, a vector encoding the kernel shapes. Crossover operation is not used in this case.

Then, the fitness of each individual in the population is evaluated by calculating the number of operations in the convolutional layers and testing the classification accuracy of the trained model on a validation set. The architecture generated by NSGA-II is trained using stochastic gradient descent (SGD), using a model training dataset. In this design, one of the fitness measures of NSGA-II is the classification accuracy which is defined as the Top-1 accuracy of trained model on the test dataset. Another fitness is the summation of MAC operations required to compute all of the convolutional layers in the network, which is calculated by (2).

Finally, NSGA-II ranks the fitness of each individual by using a non-dominated sorting approach and a diversity preservation strategy to ensure selection with elitism and a uniform spread of solutions. In non-dominated sorting, each

solution p has two entities: the first is domination count, the number of solutions that dominate p; the second is the number of solutions that p dominates. All solutions will be sorted according to each solution's domination count into multiple ranking levels. Diversity preservation is achieved by adopting a crowding distance comparison. So that, when there are two solutions with the same domination level, the one that resides in less densely populated points of the solution space is selected [33]. Following this, half of the individuals which have higher rank will be selected as the parent population for the next generation.

IV. Experimental Setup and Results

Each individual's fitness needs to be evaluated separately by training and testing the resulting network. State-of-the-art CNN architectures may require extremely large computational budgets for processing the networks. Hence, the aim in this paper is to show the improvement of the proposed method compared with conventional convolutional layers in terms of trade-off between computation costs and classification accuracy. As an example, the LeNet-5 architecture, is used here as the benchmark network to illustrate the improvement achieved by our proposed method. To evaluate the capability and the full potential for scalability of the proposed method, it is also tested on deeper CNN architectures.

A. Experimental Settings

The benchmark CNN architecture is built based on the Lenet-5 architecture [2]. The original Lenet-5 consists of two convolutional layers and two max-pooling layers, a fully-connected layer and a classification layer. In order to improve the classification accuracy of the network, we increase the number of kernels in each of the convolutional layer and nodes in the fully-connected layers. The overall architecture is shown in Fig. 5, which is used as the benchmark topology to test the proposed method.

The optimisation method is applied to three different datasets, MNIST, Fashion-MNIST and CIFAR-10. MNIST and Fashion-MNIST consist of a training set of 60,000 images and a test set of 10,000 images. Each image in the two MNIST sets is a 28×28 grayscale image, associated with a label from 10 different classes. CIFAR-10 is split into a training set of 50,000 images and a test of 10,000 images. Each image in the CIFAR set is a 32×32 pixel RGB image, associated with a label from 10 different classes. The network is trained on the training sets and evaluated in the test set, which is one of the fitness measures of the proposed method. The number of operations is calculated by the total number of multiplications in two convolutional layers, the second objective measure used here. All of the networks are trained by SGD method and use the Adam optimiser [34] with a learning rate of 1e-3. The softmax cross-entropy loss is used as the loss function. Each model is trained for 30 epochs.

In order to explore the Pareto front of the CNN architecture, the optimisation loop is set to run 100 generations for a population size of 25 individuals. There is only one type of chromosome. Hence, only mutation is used as the genetic operator with the mutation rate set to 0.1.

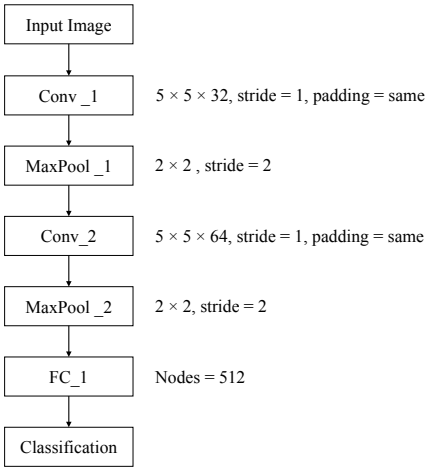

Fig. 5: The benchmark CNN used to test our optimisation method. The benchmark CNN is based on the Lenet-5 architecture. The network consists two 2-D convolutional layers, which contain 32 and 64 kernels respectively. All of the kernels have dimensions of 5×5 and the stride is 1. Each convolutional layer is followed by a max pooling one with dimensions of 2×2 and a stride of 2. There is a fully-connected layer connected to the output of the second max pooling layer that has 512 nodes. Finally, a classification layer is used to predict the best classification label applied to the image.

B. Experimental Results

The proposed method is evaluated on CIFAR-10 dataset using two optimisation objectives, number of multiplications in convolutional layers and network classification accuracy. In this experiment, the training dataset is created by randomly selecting 40,000 images from the training set, and the remaining 10,000 images are used for fitness evaluation. Finally, architectures found are trained on the training set for 100 epochs and classification accuracy is tested on test set. To prevent overfitting, a weight decay of $10e-4$ and data augmentation have been used for training the networks. The data augmentation used is based on [35], that is padding 4 pixels on each side and randomly crop a patch from the padded image or its horizontally flipped version. In order to handle the colour image inputs, the input layer of the benchmark network is modified as 3-channel input.

After configuring the benchmark network to accept colour images, the total number of multiplications required for the convolutional layers further increases to 15,564,800, and the classification error is 17.63% on CIFAR-10. The optimisation loop has a population of 25 individuals and was run for a 100 generations. The computational time of the proposed method in CIFAR-10 dataset takes about 6 days by using a NVIDIA Tesla V100. The experimental result is shown in Fig 6. Three different architectures from the set of solutions are chosen as reference points. The first reference point, Ref.

1, has the highest accuracy. The second reference point is one which involves significantly less computational resource while still featuring high accuracy. The third reference point is the trade-off solution closest to the origin between number of multiplications and classification accuracy. .

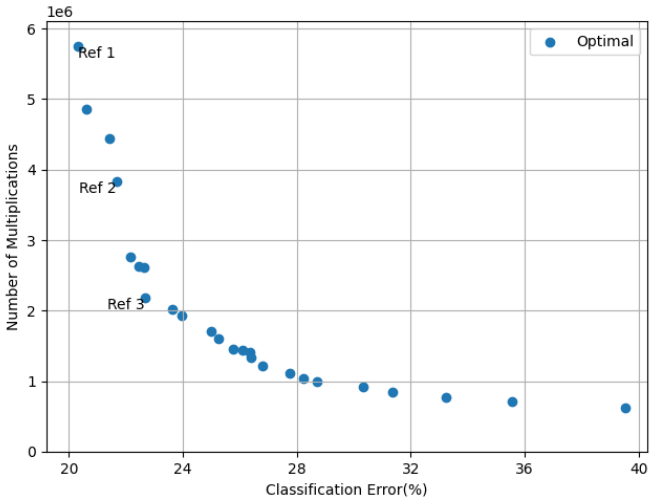

Fig. 6: The optimisation result of the proposed method after 100 generations on CIFAR-10.

Then, the proposed method is tested on MNIST and Fashion-MNIST datasets. With the same training settings, the benchmark network has a classification accuracy of 98.92% on MNIST and 98.92% on Fashion-MNIST. Both benchmark networks require a total of 10,662,400 multiplications in their convolutional layers. The computational time of the proposed method in MNIST and Fashion-MNIST datasets takes about 6 days by using a NVIDIA Tesla V100. The optimisation results are shown in fig 7. There are three reference solutions that have been selected from each set of solutions using the same approach as before: the first reference solution features the highest accuracy, the second reference solution uses less resources while featuring high accuracy, and the third reference solution has the closest-to-origin trade-off between number of multiplications and classification accuracy.

Three architectures from each dataset are then re-trained on the whole training set and the classification accuracy is evaluated on the test set which contains 10,000 examples. Each architecture is re-trained for 100 epochs with weight decay and data augmentation. All of them are trained by using Adam optimiser [34] with initial learning rate of 0.001 and the learning rate is reduced by factor of 10 at 30th epoch. After re-training and testing, the results are shown on Table I. The comparison between the benchmark network and reference point is shown in Table I.

V. Conclusion and Further Work

In this paper, we proposed a generic Multi-objective Evolutionary Algorithm (MOEA)-based approach for optimising the size and efficiency of CNN architectures by introducing unconventional (non-square) kernel shapes and combining

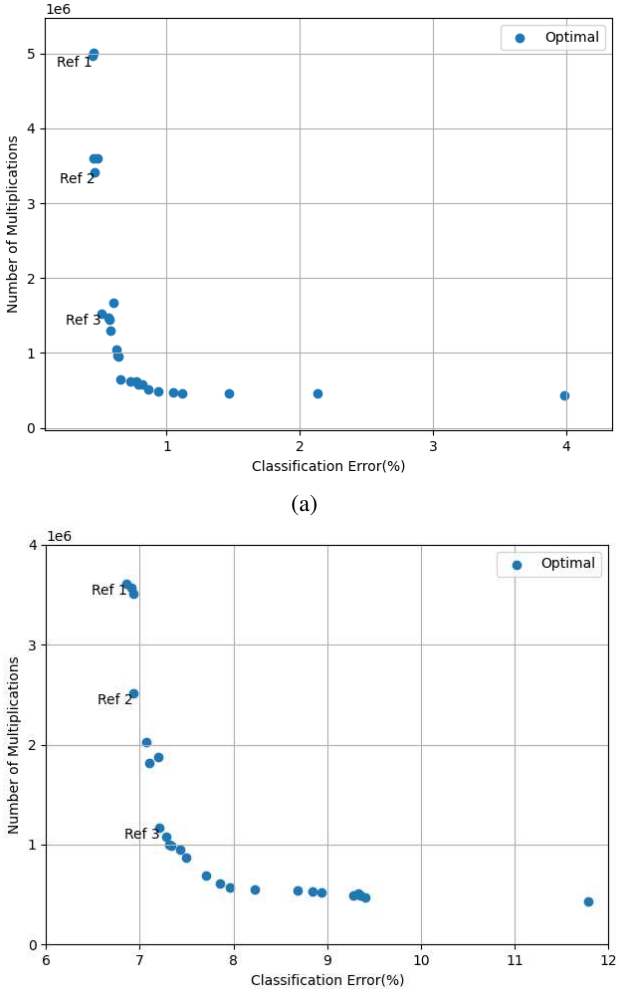

Fig. 7: (a) The optimisation result of the proposed method after 100 generations on MNIST. (b) The optimisation result of the proposed method after 100 generations on Fashion-MNIST.

TABLE I: Comparison between the benchmark network and solutions found by the proposed method on CIFAR-10, MNIST and Fashion-MNIST datasets. Three reference points are selected from optimised results for each dataset.

Dataset	Model	Accuracy	Acc. improve	Reduction in Mults.
MNIST	Benchmark	98.92%	-	-
	Ref 1	99.56%	0.64%	2.15x
	Ref 2	99.54%	0.62%	3.13x
	Ref 3	99.49%	0.57%	7.02x
Fashion-MNIST	Benchmark	92.12%	-	-
	Ref 1	93.14%	1.02%	2.95x
	Ref 2	93.07%	0.95%	4.24x
	Ref 3	92.79%	0.67%	9.15x
CIFAR-10	Benchmark	82.37%	-	-
	Ref 1	83.75%	1.38%	2.71x
	Ref 2	82.54%	0.17%	4.06x
	Ref 3	80.84%	-1.53%	7.14x

different sizes of convolution kernels. The proposed method automatically generates combinations of these unconventional kernels that are used to replace the set of one-size square

convolution kernels produced by a conventional approach. The optimisation by MOEA provides a trade-off solution space between computational resources and classification accuracy, which is unique to such algorithms. The results show that a significant reduction in the computational resource consumption with negligible sacrifice of (and sometimes slightly increased) performance.

The proposed method has been tested on commonly-used benchmarks, MNIST, Fashion-MNIST and CIFAR-10 datasets with CNN architectures. As can be seen from the results, the proposed method achieves significant improvements in computational resource consumption, sometimes with increases in classification accuracy, over conventional designs using square kernels only. Trading off accuracy with computational complexity of resource consumption in CNNs running in resource-constrained environments is a real-world problem. The significant reduction of computational resources achieved here allows deep CNN architectures to be more efficiently implemented on many resource-constrained platforms, such as Field Programmable Gate Arrays (FPGAs) and embedded devices. The methodology of adapting kernel shapes shows promise to be further generalised in the future, particularly with new devices featuring relevant hardware to execute, e.g., sparse matrix multiplications, efficiently.

REFERENCES

[1] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, and M. S. Lew, "Deep learning for visual understanding: A review," *Neurocomputing*, vol. 187, pp. 27–48, 2016.

[2] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[3] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks." *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 6, p. 1137, 2017.

[5] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[6] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv preprint arXiv:1511.07122*, 2015.

[7] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. on Comp. Vision and Pattern Recog.*, 2015, pp. 1–9.

[8] 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.

[9] A. Canziani, A. Paszke, and E. Culurciello, "An analysis of deep neural network models for practical applications," *arXiv preprint arXiv:1605.07678*, 2016.

[10] C. A. C. Coello, "A short tutorial on evolutionary multiobjective optimization," in *International Conference on Evolutionary Multi-Criterion Optimization*. Springer, 2001, pp. 21–40.

[11] L. Xie and A. Yuille, "Genetic cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 1379–1388.

[12] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proc. of the Genetic and Evolutionary Comp. Conf.*, 2017, pp. 497–504.

[13] D. E. Moriarty and R. Miikkulainen, "Hierarchical evolution of neural networks," in *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98TH8360)*. IEEE, 1998, pp. 428–433.

[14] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[15] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, vol. 2, 2010.

[16] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[17] A. Krizhevsky and G. Hinton, "Convolutional deep belief networks on cifar-10," *Unpublished manuscript*, vol. 40, no. 7, pp. 1–9, 2010.

[18] K. O. Stanley and R. Miikkulainen, "Efficient evolution of neural network topologies," in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, vol. 2. IEEE, 2002, pp. 1757–1762.

[19] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, "A hypercube-based encoding for evolving large-scale neural networks," *Artificial life*, vol. 15, no. 2, pp. 185–212, 2009.

[20] P. Verbancsics and J. Harguess, "Image classification using generative neuro evolution for deep learning," in *2015 IEEE winter conference on applications of computer vision*. IEEE, 2015, pp. 488–493.

[21] G. Morse and K. O. Stanley, "Simple evolutionary optimization can rival stochastic gradient descent in neural networks," in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, 2016, pp. 477–484.

[22] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 2902–2911.

[23] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy *et al.*, "Evolving deep neural networks," in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Elsevier, 2019, pp. 293–312.

[24] Y.-H. Kim, B. Reddy, S. Yun, and C. Seo, "Nemo: Neuro-evolution with multiobjective optimization of deep neural network for speed and accuracy," in *JMLR: Workshop and Conference Proceedings*, vol. 1, 2017, pp. 1–8.

[25] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Automatically designing cnn architectures using genetic algorithm for image classification," *preprint arXiv:1808.03818*, 2018.

[26] J. F. Miller and S. L. Harding, "Cartesian genetic programming," in *Proc. of the 10th annual conf. companion on Genetic and evolutionary computation*, 2008, pp. 2701–2726.

[27] A. Sironi, B. Tekin, R. Rigamonti, V. Lepetit, and P. Fua, "Learning separable filters," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 1, pp. 94–106, 2014.

[28] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," *preprint arXiv:1405.3866*, 2014.

[29] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Advances in neural information processing systems*, 2014, pp. 1269–1277.

[30] J. Jin, A. Dundar, and E. Culurciello, "Flattened convolutional neural networks for feedforward acceleration," *preprint arXiv:1412.5474*, 2014.

[31] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. of the IEEE conf. on comp. vision and pattern recognition*, 2016, pp. 2818–2826.

[32] X. Ding, Y. Guo, G. Ding, and J. Han, "Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks," in *Proc. of the IEEE Int. Conf. on Computer Vision*, 2019, pp. 1911–1920.

[33] C. C. Coello, "Evolutionary multi-objective optimization: a historical view of the field," *IEEE computational intelligence magazine*, vol. 1, no. 1, pp. 28–36, 2006.

[34] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *preprint arXiv:1412.6980*, 2014.

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