Evolutionary Optimisation in Convolutional Neural Networks

Ziwei Wang

Doctor of Philosophy

University of York

Electronic Engineering

September 2022

Abstract

Artificial Neural Networks (ANNs) have increased in performance in recent years and have been successfully used in many computer applications. The growth in ANNs and machine learning applications has resulted in the state-of-the-art ANNs comprising hundreds of hidden layers, which require millions of operations to process input data and gigabytes of memory to store model parameters. The complexity of ANNs limits them to be implemented on resource-constrained platforms for certain applications, such as Internet-of-Things (IoT). Therefore, it is obvious to try and optimise ANNs so that the computational cost and model size of ANNs is reduced, while maintaining high accuracy in classification. Evolutionary algorithms demonstrate flexible capabilities for solving optimisation problems with one or more objectives. Hence, applying evolutionary algorithms to optimise ANNs becomes a potential solution.

In this PhD work, evolutionary techniques are applied to the optimisation of ANNs, which aims to reduce the computational cost and parameter size while minimising loss in classification accuracy. The optimisation is divided into two categories, i.e. computational cost optimisation, and data representation optimisation. For the computational cost optimisation, a multi-objective evolutionary approach is proposed to reduce the size and number of convolution kernels in each convolutional layer and generate trade-offs between computational cost and model's classification accuracy. For data representation optimisation, an evolutionary-based adaptive integer quantisation methodology is introduced to quantise the pre-trained models from 32-bit floating point representation to small bit-width integer representation.

The experimental results for computational cost reduction that multi-objective evolutionary algorithms achieve large improvements in resource consumption with no significant reduction in a model classification accuracy, compared with the original models' architecture. From the experimental results on data representation optimisation, evolutionary-based adaptive integer quantisation methodology illustrates that weights and biases in convolutional layers can be quantised to 8-bit integer representation and 4-bit integer representation in fully-connected layers without significant gap in models' classification accuracy between pre-trained 32-bit floating point representation and quantised weights and biases.

Table of Contents

List of Tables

List of Figures

Acknowledgements

My most sincere and heartfelt gratitude goes to my supervisors Professor Andy Tyrrell and Dr. Martin Trefzer. Thanks for all of their tireless guidance and support throughout my four-year PhD life. It is my honour to be supervised by them.

Secondly, I would like to thank my parents for their love, support and understanding during my PhD life.

Declaration

I declare that this thesis is a presentation of original work and I am the sole author. This work has not previously been presented for an award at this, or any other, University. All sources are acknowledged as References.

Publications

- Z. Wang, M. A. Trefzer, S. J. Bale, and A. M. Tyrrell, "Approximate multiply-accumulate array for convolutional neural networks on fpga," in *2019 14th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*. IEEE, 2019, pp. 35–42

- Z. Wang, M. A. Trefzer, S. Bale, and A. M. Tyrrell, "Adaptive integer quantisation for convolutional neural networks through evolutionary algorithms," in *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2021, pp. 1–7

- Z. Wang, M. A. Trefzer, S. J. Bale, and A. M. Tyrrell, "A multi-objective evolutionary approach for efficient kernel size and shape for cnn," in *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2022, pp. 1–8

Ziwei Wang

September 2022

Chapter 1

Introduction

Deep Neural Networks (DNNs) are biologically-inspired computing systems which consist of internal connections between artificial neurons. In recent years, DNNs have increasingly drawn attention in various application domains, such as image processing, speech recognition and many other challenging computational tasks [4]. In particular, Convolutional Neural Networks (CNNs) have successfully gained outstanding performance in image classification and video recognition. However, while state-of-the-art CNNs have become increasingly accurate, these networks are computationally expensive involving billions of arithmetic operations and parameters [5–7]. Hence, the development of CNNs raises big challenges related to apply these techniques resource-constrained systems.

This thesis explores the design methodologies of CNNs [5–7] with a specific focus on analysing the computational resource consumption and parameter set size require to process CNN models. This work will focus on investigating how the computational resource consumption and parameter size of CNNs models can be optimised by Multi-objective Evolutionary Algorithms (MOEAs). This work will not only aim to reduce the costs of processing CNNs, but also to retain the classification accuracy of optimised CNNs models as much as possible.

1.1 Motivation

In recent years, the state-of-the-art achievements of designing CNN architectures, e.g. GoogleNet [7], VGG [5] and AlexNet [6], have demonstrated significant improvements of classification accuracy on various benchmark datasets, such as MNIST [8], CIFAR [9] and ImageNet [10]. The general methodology for designing highly-accurate CNN models is making the CNN deeper and wider, i.e models contain increasingly large numbers of layers, as well as more neurons in each each layer. However, as the depth

and width of these models grows larger, the computational complexity and parameter size of the network grows exponentially.

Current trends in edge computing require implementation of CNNs in low-cost embedded devices and resource-constrained platforms. For certain applications, such as Internet-of-Things (IoT), the CNNs are required to process with limited computational resources and memory size. However, such applications aiming at resource and memory-constrained devices suffer from limitations, i.e. computational resources and memory size, making large CNNs difficult to process with sufficient performance. To deal with this problem, recent research has shown the potential of providing high-accuracy and low-cost CNN models [11–13]. The most significant problem that limits the computational speed is that processing CNNs requires massive computational resources for multiply-accumulation. In order to achieve high accuracy, state-of-the-art CNN models consist of many hidden layers. These hidden layers can significantly increase the model accuracy. However, as the networks become deeper, a huge amount of operations and processing data are produced. For example, the VGGNet has a total of 16 layers and consists of 138 million parameters [5] and storage requirement of network and parameters is up to 552 MB [14]. Therefore, the motivation to optimise existing models that can achieve high accuracy while keeping computational resource consumption at a minimum is therefore high.

Evolutionary algorithms (EAs) are today one of the most commonly-used methodologies for solving hard optimisation problems where one or more conflicting objectives must be satisfied simultaneously [15–18]. EAs aim to generate a set of possible solutions (so-called population) in a single run of the algorithm. In recent years, several research studies have been 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 [19–21].

Overall, the proposed research in this PhD thesis is motivated by reducing the computational and memory costs while retaining high accuracy when CNN models are in place for specific tasks. The proposed research focuses on applying bio-inspired techniques to apply multi-objective optimisation methodologies for reducing computational and memory consumption of existing CNN models, enabling trade-off solutions for applying CNNs models in different use case scenarios. Multi-objective evolutionary algorithms are potentially well-suited for optimising complex CNN models, as they can consider both computational costs and classification accuracy at the same time from a global viewpoint.

1.2 Hypotheses and Objectives

The research presented in this thesis proposes and evaluates how resource and memory consumption can be optimised by evolutionary algorithms based on following hypotheses:

Hypothesis: Applying evolutionary algorithms to optimise structure of trained CNN models can achieve significant improvements in resource consumption and memory usage while maintaining the classification accuracy of the original models via reducing the number of operations required for processing convolutional layers, and applying low-precision integer data representation for trained CNN models' parameters, i.e. weights and biases.

Based on this, the following sub-hypotheses can be formula led:

Sub-hypothesis 1: Unconventional shapes of convolution kernels, e.g. 1×3 and 3×1 kernels, can be used to replace some of the commonly-used square convolution kernels to reduce the computational cost of convolutional layers.

Sub-hypothesis 2: Multi-objective evolutionary algorithms can be used to optimise the computational resource consumption by reducing the size, shape and number of kernels in convolutional layers for specific tasks.

Sub-hypothesis 3: Evolutionary algorithms are capable of quantising CNN weights and biases from their original 32-bit floating point representation to small bit-width integer representation while minimising the loss in classification accuracy.

Sub-hypothesis 4: Applying different bit-widths of integer representation for convolutional layers and fully-connected layers can achieve further reduction in parameter size while minimising the loss in classification accuracy.

In order to test these hypotheses, the following objectives will be met:

Objective 1: Develop an optimisation methodology to reduce computational cost of processing feed-forward CNNs with combinations of various unconventional convolution kernel shapes and sizes.

Objective 2: Devise a methodology that allows adaptive quantisation of both weights and bias from 32-bit floating point representation to lower bit-width integer representation while retaining model classification accuracy.

1.3 Contributions

This thesis makes several contributions to the field of optimising CNN models. These contributions can be divided into two main categories, computational cost optimisation and data size optimisation. Details for contributions in each categories have been described as follows:

For computational cost optimisation perspective:

- A framework of multi-objective optimisation is proposed to explore the design space for convolution kernels and produce the trade-off between neural network computational consumption and classification accuracy.
- A methodology introducing unconventional (non-square) kernel shapes and combining different sizes of convolution kernels, which 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 knowledge of how different shapes and sizes of convolution kernels are affecting the computational resource consumption and classification accuracy when implementing CNNs.

For data representation optimisation perspective:

- The creation of an evolutionary algorithm based adaptive integer quantisation methodology that quantises both weights and biases of pre-trained CNN models from their original 32-bits floating point representation to smaller bit-width integer representation, while keeping the classification error at a minimum after quantisation.
- A methodology that applies evolutionary algorithm to find upper and lower boundary for each quantisation bin within the low precision integer representation for a given pre-trained CNN weights and bias.
- The knowledge of how different bit-widths of integer representation affect the parameter size and classification accuracy, and how the model accuracy is affected when applying different integer representation for quantising convolutional layers and fully-connected layers of pre-trained CNN models.

1.4 Thesis Structure

This thesis consists of seven chapters and is organised as follows:

- Chapter 2 provides a background review of recent developments in CNN designs and different approaches of optimising CNNs, as well as recent approaches of applying evolutionary algorithms for searching neural network architectures.
- Chapter 3 presents a computational resource consumption optimisation to trade-off between the amount of computation and model classification accuracy. A MOEA is applied together with multiple unconventional (non-square) convolution kernels.
- Chapter 4 extends the computation consumption optimisation that considers allowing the computation consumption optimisation to remove some convolution kernels, if possible, with the aim to further reduce the computational costs in convolutional layers.
- Chapter 5 demonstrates an adaptive integer quantisation approach for data representation optimisation that applies a rank-based evolutionary strategy to reduce the number of bits by quantising pre-trained CNN models from 32-bit floating point representation to 8-bit integer weights and bias.
- Chapter 6 extends the adaptive integer quantisation to apply different integer representation in convolutional layers and fully-connected layers for further reducing the parameter size of CNN models.
- Chapter 7 concludes all frameworks and methodologies for optimising CNN models that have been proposed in this thesis, make suggestions for improvements, and further works are discussed to further explore current findings.

Chapter 2

Background

This chapter provides the background knowledge of artificial neural networks and evolutionary algorithms, as well as a review of recent approaches looking at evolutionary optimisation techniques that have been developed for optimising artificial neural networks.

2.1 Convolutional Neural Networks

In this section, the general principle of Convolutional Neural Networks (CNNs) is introduced. In addition, a review of recent research approaches for CNN architecture design is provide.

2.1.1 Introduction to Convolutional Neural Networks

Artificial Neural Networks (ANNs) are biologically-inspired computing systems which consist of inner-connections between artificial neurons. ANN models aim to build a correspondence between their inputs and outputs by adjusting parameters attached to neurons. ANN systems aim to perform tasks by training them on data, they are not initially designed for specific applications. ANN computing systems are learning to perform tasks by considering training data, generally without being programmed with any task-specific rules. Therefore, this less-specific feature allows ANNs to be used in various application areas. Furthermore, due to their non-specific nature, an ANN model can be adapted to different applications with less effort compared with conventional mathematical algorithms. These advantages make ANNs an attractive topic in both academic and industrial research.

An ANN model is often used for Deep Learning (DL) tasks, called Deep Neural Networks (DNNs). Deep learning is a branch in the wider family of Machine Learning. It is designed for automatically evaluating the relationship, features or representations,

from the training dataset without specific human designs of feature extractions. Bengio et al. [22] introduced deep architectures composed of multiple levels of non-linear operations, such as in neural nets with many hidden layers. Accordingly, A DNN model usually consists multiple hidden layers, a distributed representation between input data and its outputs, and non-linear activation functions. In recent years, many DNN models have been developed for adapting them to different applications, such as Multi-layer Perceptron (MLP) and Convolutional Neural Networks (CNNs).

Multi-Layer Perceptron

The Multi-Layer Perceptron (MLP) is also known as Fully-Connected Neural Network (FCNN). A Fully-Connected Neural Network (FCNN) means that an output neuron connects to every input neuron in the previous layer, as shown in Fig. 2.1.

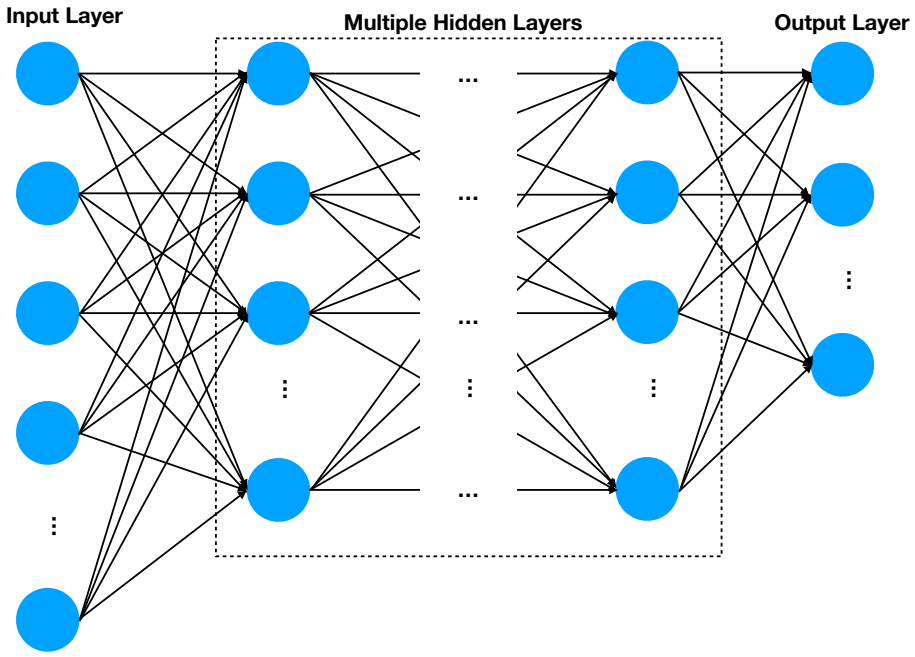


Figure 2.1 A feed-forward Fully-Connected Neural Network

Fig. 2.1 shows the general architecture of a feed-forward Fully-Connected Neural Network. The feed-forward neural network is using pre-trained weights to predict

the category of new input data. While processing an artificial neural network, a vector-matrix-multiply operation is involved to compute outputs for each neuron. This can be broken down into a series of multiply-accumulate (MAC) operations [23]. A feed-forward neural network usually contains many layers of neurons (deep nets). Each layer receives data from the previous layer as its input. There are normally many neurons in each layer. Every neuron also contains a weight and bias to each of its inputs. The sum of bias and weighted inputs are subject to an activation function which is used to limit the output of neurons to a specific range, as described in (2.1) and Fig. 2.2.

$$O_i = g\left(\sum_{j=1}^{n} w_{ij}x_j + b_i\right) \tag{2.1}$$

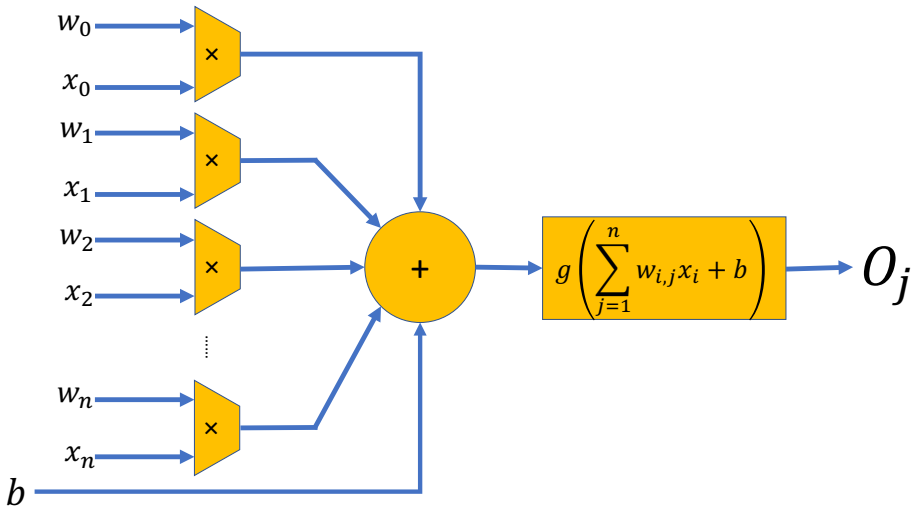


Figure 2.2 Data flow of neurons.

As shown in (2.1) and Fig. 2.2, the multiply-accumulate result is then calculated through a non-linear transformation, known as activation function $g(x)$. Without the activation function, the value of MAC operation can be anything ranging between negative infinite and positive infinite and the multi-layer neural network would simply collapse into a single linear equation. Hence, the non-linear activation function, the output feature would simply be the linear combination of input features. Therefore, an activation function with nonlinearity has been added to each layer.

In recent years, more and more DNNs tended to use rectified linear units (ReLU) as activation functions. The ReLU function can be divided into two linear segments: (1) Outputting zero if the input is negative. (2) The output value is equal to the input value if the input is positive. The ReLU function makes the network avoid problems stemming from saturating excitation, and keeps gradients from vanishing for deep networks [24]. From a hardware perspective, ReLU is also easier to implement, with less overhead, than other activation functions, such as *sigmoid* and *tanh*. The reason is that a negative input causes the ReLU to output 0, and a positive input gives the output as $f(x) = x$. Therefore, it can be simply processed with a sign-detector and a multiplexer in digital circuit implementation. Table 2.1 lists some commonly used activation functions in ANNs.

Table 2.1 Commonly used activation functions in artificial neural networks.

Type	Function
ReLU	$f(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$
Leaky ReLU	$f(x) = \begin{cases} x & x \geq 0 \\ 0.01x & x < 0 \end{cases}$
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$
Tanh	$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
Softmax	$f(x) = \frac{e^{x}}{\sum_{j=1}^{J} e^{x}}$

Convolutional Neural Networks

Compared with other types of neural networks, a convolutional neural network consists of different types of layers, including 2-D convolutional layers, pooling layers and fully-connected layer. The 2-D convolutional layers and pooling layers have several feature maps. These feature maps are used to detect different features from the input

data, and pooling layers are used to summarise the presence of those features from previous convolutional layers. The detected features then pass through the classification layer which is used to select appropriate responses for those features. When compared to a fully-connected neural network, convolutional layers usually contain smaller kernel sizes. Neurons in the convolutional kernel are shifting over the input data by using the same set of weights and bias. Then, the sum of weighted inputs will be input to the activation function and the outputs from the activation function are used as the inputs of the next layer, as shown in Fig. 2.3.

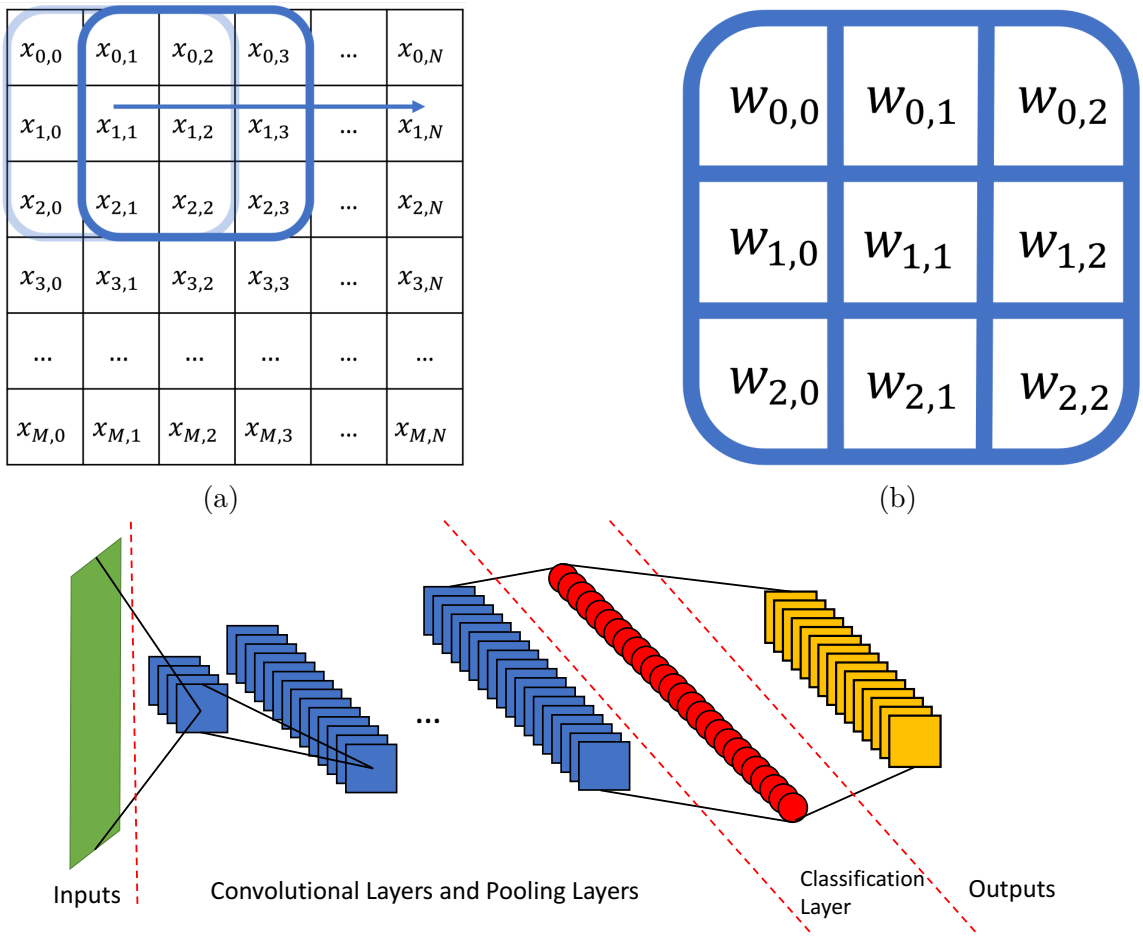


Figure 2.3 (a) An example of shifting 3x3 convolutional kernel across the image. (b) 3x3 convolutional kernel. (c) Example of the whole CNN process.

The convolutional layers and pooling layers in CNNs are sparsely connected. Compared with fully-connected layer in feed-forward neural networks, neurons in a convolutional layer are only connected with a few neurons in previous layer, as apposed to all of the neurons. 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 convolutional kernel in the previous layer [23, 25]. The sparse connectivity of a CNN has a regularisation effect, which improves the stability and generalisation ability of the network structure as well as avoiding overfitting. Meanwhile, sparse connection allows networks with reduced number of weight parameters, which is conductive to the fast learning of a neural network and reducing memory footprint while processing the network [26].

Since the AlexNet achieved first place in ILSVRC-2012 competition in 2012 [6], many researchers started to focus on designing more efficient and accurate CNN architectures. In the remainder of this section, some of the most widely used CNN architectures are reviewed, including LeNet-5, AlexNet and etcs.

LeNet-5 was proposed by LeCun et al [27]. It has been widely used in the United States for automatically classifying handwritten digits on bank cheques. Before LeNet-5 was proposed, character recognition was achieved by feature extraction through manual feature engineering, then, applied machine learning models to learn the extracted features for object classification. Feature engineering was a big issue, that engineers had to consider what kind of features were necessary. The LeNet-5 could extract and learn features by itself. This mean that it was possible for the network model to learn features by itself, and does not require specific manual design for feature extraction methods.

LeNet-5 contains an input layer with 32×32 neurons which are used to handle grey-scale input images. After the input layer, LeNet-5 contains three convolutional layers that are built up by 6 and 16 5×5 and 120 1×1 convolution kernels respectively. First two convolutional layers are followed by an average pooling operation. There is

a fully-connected layer with 84 neurons connected to the last average pooling layer. Finally, a ten-neuron fully-connected classification layer is connected at the end to classify 0 to 9 handwritten digits. The LeNet-5 is the pointer of convolutional neural network that introduced the local receptive field, shared weights and subsampling.

AlexNet is the dividing line between shallow neural networks and deep neural networks which won the ILSVRC-2012 image recognition competition [6]. The AlexNet contains five convolutional layers, three pooling layers and three fully-connected layers. AlexNet has similar architecture to LeNet-5, but it is much deeper and wider than LeNet-5 which involves larger parameter space to fit large-scale datasets. The AlexNet has five convolutional layers, three maxpooling layers and three fully-connected layers. Compared with LeNet-5, the AlexNet contains four key components:

1. AlexNet uses $ReLu$ [28] as the activation function, instead of $Sigmoid$ or $Tanh$ activate functions which were conventionally used in previous CNNs designs. The reason for applying $ReLu$ as the activation function is that calculating the derivative of $Sigmoid$ and $Tanh$ is cumbersome, and the gradient of both $Sigmoid$ and $Tanh$ will disappear when the CNN becomes deeper.

2. AlexNet implements Local Response Normalisation (LRN) into the network. The LRN creates a competition mechanism for the activity of local neurons, making the values in which the response is relatively large and suppressing other neurons with smaller feedback, then, enhancing the generalisation ability of the model.

3. AlexNet applies max pooling operation to the output of convolutional layers. Previously, average pooling was commonly used in CNNs, AlexNet uses max pooling in all pooling layers to avoid the blurring effect of average pooling. It is proposed in AlexNet to let the step length be smaller than the size of the pooling kernel, so that there will be overlap and coverage between the outputs of pooling layers, which improves the feature richness.

4. Dropout [29] is used when training AlexNet, randomly ignoring some neurons in order to avoid overfitting the network. It is mainly the last few fully connected layers in AlexNet that use Dropout.

VGGNet was proposed by K.Simonyan and A.Zisserman in 2014. It won the second place in ILSVRC-2014 [5]. Until now, VGGNet is still frequently used to extract image features in various applications. VGGNet explores the relationship between the depth of a CNN and its performance. By iteratively stacking small convolution kernels, 3×3, and max pooling size of 2×2, VGGNet successfully constructs 11-layers to 19-layers deep CNNs. Literature suggests that the increased depth of the convolutional neural network and the use of small convolution kernels in VGGNet play an important role in the network's classification and recognition accuracy. The Fig 2.4 [5] shows different network configurations of VGGNet from 11-layers to 19-layers structures. As is shown in Fig 2.4, each convolutional layer, such as $conv3$, contains multiple convolution kernels of size 3×3 with stride of 1 and padding 1, so that the height and width of the feature maps are not changed after the convolution operation. The maxpooling layers contain pooling kernel size of 2 and a stride of 2. The height and width of the feature maps are reduced to half of the input size after each pass through the maxpooling operation. The $ReLu$ activations are applied to the outputs of each layer, except the last fully-connected, FC-1000, where $Softmax$ is placed.

VGGNet uses the following specifications:

1. The network structure of VGGNet is very simple. It consists of five convolutional layers, three fully-connected layers and one softmax layer. The layers are connected to each other using maxpooling and the $ReLu$ activation functions used between all hidden layers.

2. VGGNet uses a convolutional layer that contains multiple small 3×3 convolution kernels instead of large kernels like AlexNet [6]. The receptive field of a stack of

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 2.4 Different configurations of VGGNet. In this table, the "11 weights layers" means there are 11 layers in total (excluding maxpooling layers). "LRN" means Local Responds Normalisation. The "conv3-64" means there are 64 3×3 convolution kernels in current layer, "conv3-128" means there are 128 3×3 convolution kernels in current layer and so on. "FC-4096" means a fully-connected layer that contains 4096 neurons.

two 3×3 convolution kernels is equivalent to the receptive field of a 5×5 kernel, and the receptive field of a stack of three 3×3 convolutional kernels is equivalent to the perceptual field of a 7×7 kernel which is used in AlexNet. Thus, the use of multiple small convolution kernels can not only reduce the number of parameters, but also enhances the nonlinear mapping of the networks, thus, improving the expressiveness of the network.

3. VGGNet has a large number of channels in each layer. The first layer of VGGNet has 64 channels and each subsequent layer doubles the number of channels to a maximum of 512 (64-128-256-512-512). Since each channel represents a feature map, this allows more information to be extracted.

GoogleNet was developed by C. Szegedy et al in 2014, it won the first place in the Classification Task of ILSVRC-2014 [7]. Unlike previous CNN designs, e.g. AlexNet and VGGNet, where only a single size of convolution kernels is placed in each convolutional layer, GoogleNet introduced a new model of convolutional layer which is defined as 'Inception'. The structure on inception module is shown in Fig 2.5 [7].

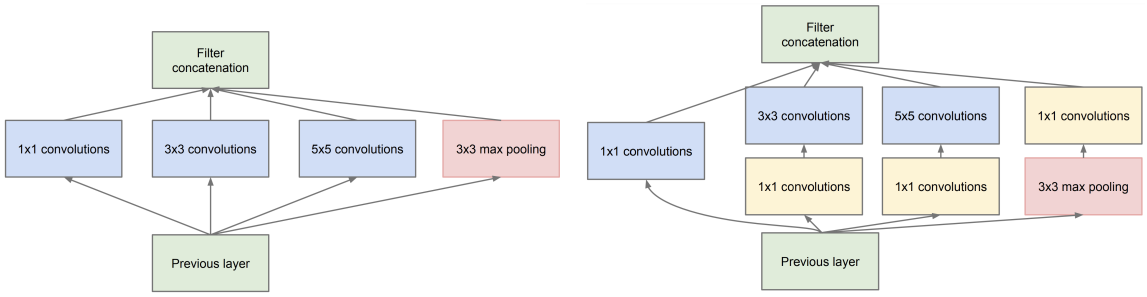


Figure 2.5 (a) The initial version of Inception module (b) The dimension reduction version of Inception module. An Inception module consists of multiple size of convolution kernels in parallel. There are some 1×1 kernels in the Inception module before passing them to the parallel operations, which are used to reduce the depth of tensors.

The Fig 2.5a demonstrates the initial version of Inception module in GoogleNet. Unlike the previous CNNs architecture designs, the convolutional layers were connected in series with each other and with the pooling layers, whereas in GoogleNet, they are connected in parallel with multiple size and types of convolution kernels. The Inception module shown in Fig. 2.5b adds three sets of 1×1 convolution kernels, which reduces the depth of the feature maps. As a result, these sets of 1×1 convolution kernels are greatly reducing the number of model parameters and, hence, are reducing the computational costs for processing such networks. Furthermore, GoogleNet drops the

fully-connected layers in favour of an average pooling layer which significantly reduces the model parameters.

ResNet was proposed by K. He et al [30] in 2015, it won the first place in the Classification Tasks in ILSVRC-2014 and first place in target detection and image segmentation in the COCO 2015 competition [31]. Before the ResNet was proposed, all CNNs were composed of a combination of convolutional and pooling layers. Previous research suggested that the more convolutional and pooling layers, the more features can be extracted from images, thus, the better the learning performance would be. However, in practical experiments, it was found that, as the number of convolutional and pooling layers were added, instead of getting better classifications, two problems occurred, i.e. 1) vanishing gradient and exploding gradient and 2) degeneracy. *Vanishing gradient* means that, if the error gradient at each layer is less than 1 while processing the back propagation, the gradient will converge close to 0 when the network goes deeper. The *exploding gradient* means that, if the error gradient at each layer is greater than 1 while processing the back propagation, the gradient will become higher and higher as the network becomes deeper. Degeneracy means that, as the number of layers increases, the prediction accuracy of network is getting worse. In order to deal with the vanishing gradient and exploding gradient, the ResNet introduces Batch Normalisation (BN) layers in the network to solve the issue. For mitigating the degeneracy problem, ResNet introduces 'residual blocks' to handle the issue.

- The residual structure is a kind of shortcut connection. For a stacked layer structure, the learned features are noted as $H(x)$ when the input is x, now we want it to learn the residuals $F(x) = H(x) - x$, so that the original learned features are actually $F(x) + X$. This is because the residual learning is easier than direct learning of the original features. If the network goes deeper and there is a degradation problem occurred, then we only need to make the residual map F(x) equal to 0, that is, the map H(x) to be solved is equal to the feature map x output by the previous layer, because x is the best solution for the current

output, so that the network state of the residual block is still the best one. Fig. 2.6 shows the comparison between the stacked layer and residual structure. This feature makes that the residual map will generally be small, therefore, the network requires less learning effort. When the residuals are 0, the stacking layer is only doing constant mapping at this point, so that the network performance does not degrade at least. In reality, the residual cannot be 0. This will also allow the stacking layer to learn new features based on the input feature maps, thus, achieving better performance.

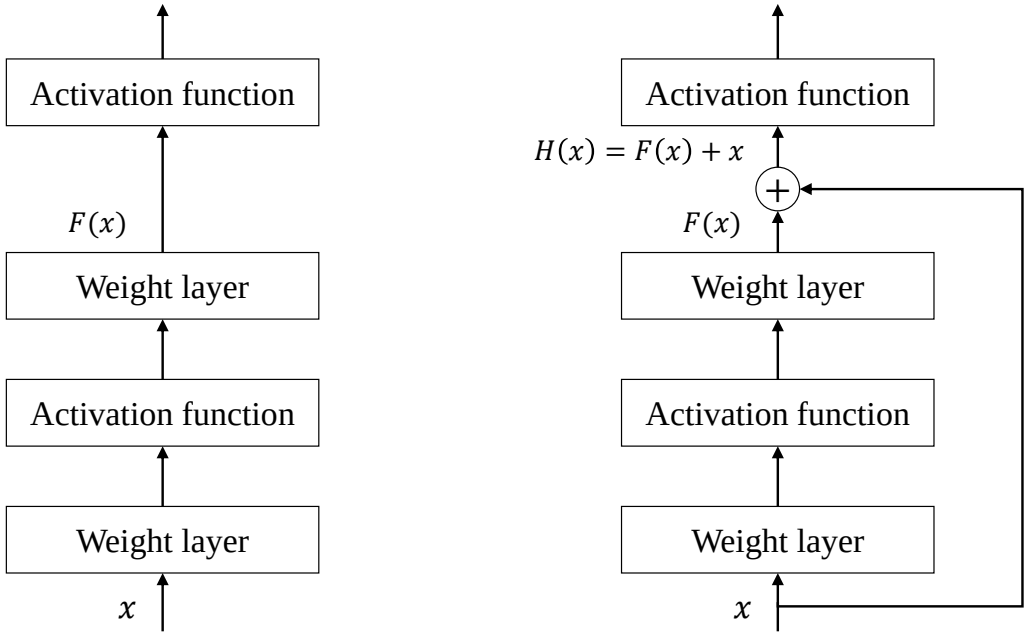


Figure 2.6 The figure in the left shows a stacked layer structure, where the weight layer needs to learn the mapping $F(x)$ directly. The figure in the right demonstrates the residual structure. In this case, the weight layer needs to learn the residual mapping $F(x) = H(x) - x$.

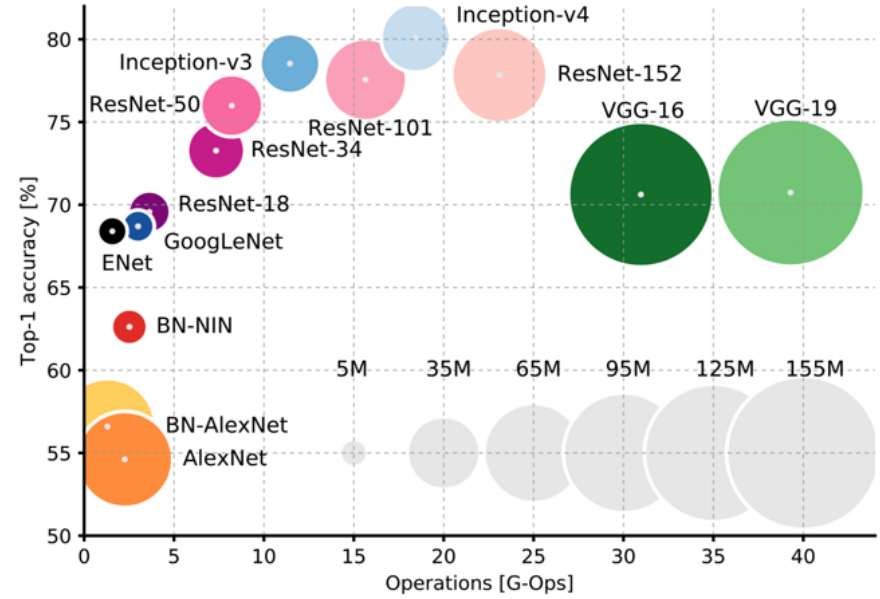
- Batch Normalisation is the process of taking a feature map of a batch of data and scaling it with a mean of 0 and a variance of 1. This process will accelerate the convergence of the network. In the training process, the CNN is trained by taking data from one batch to another. Therefore, it is required to calculate the mean and variance of each batch continuously during the training process, then record the statistical mean and variance using the moving average method.

So that, the statistical mean and variance can be approximated to be equal to the mean and variance of the whole training set after each epoch. The mean and variance of the statistics are then normalised for use in the validation and prediction process [32]. The (2.2) describes the mathematical approach for the batch normalisation, where m is the number of inputs in the mini-batch, $\mathcal{B}$ refers to the current batch, $\mu_{\mathcal{B}}$ is the batch mean, $\sigma^2_{\mathcal{B}}$ is the input variance of current batch, $\hat{x}_i$ indicates the normalised layer input with the batch mean and variance, ϵ is for numerical stability in case the denominator turns to zero and y_i is the output of the layer after scaling and shifting. The scaling and shifting parameters, γ and β are learned during the training process.

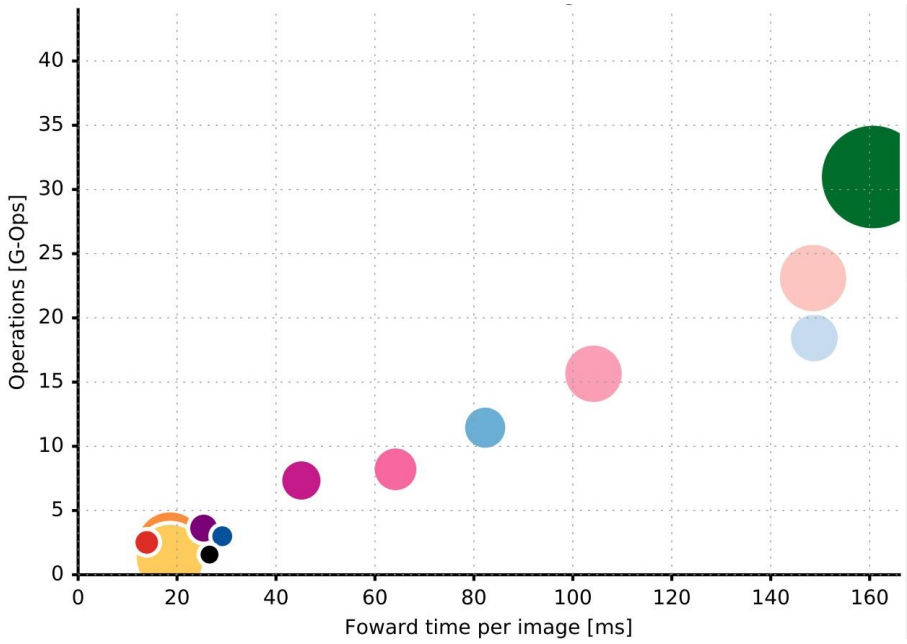
$$\begin{aligned} \mu_{\mathcal{B}} &= \frac{1}{m}\sum_{i=1}^{m} x_i \\ \sigma^2_{\mathcal{B}} &= \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \\ \hat{x}_i &= \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \\ y_i &= \gamma\hat{x}_i + \beta \end{aligned} \tag{2.2}$$

2.1.2 Computational and Memory Cost Analysis

This section provides a review of some classic CNN architectures. From a global point of views for different CNN architectures, the Fig. 2.7a illustrates the trend between top-1 classification accuracy, number of operations for processing input data for corresponding CNNs and parameter size of recent developed CNNs on ImageNet [13]. It can be seen from the figure, as the classification accuracy of CNN goes higher, the network requires more computational resources to process input data and a larger memory size to store model's parameters. In order to achieve high accuracy, state-of-the-art ANN models consist of many hidden layers. These hidden layers can significantly increase the model accuracy. However, as the networks goes deeper, a huge amount of



(a)



(b)

Figure 2.7 (a) Classification Accuracy vs Operations vs Model Size of recent CNNs on ImageNet. (b) Forward time per image vs Operations for a single input image. In both figures, each CNN is represented by an unique colour.

operations and processing data are produced. For example, the VGGNet has a total of 16 layers and consists of 138 million parameters [5] and storage requirement is up

to 552 MB [14] and the Caffe [33] version of ALexNet takes 106.09ms to process for single 224×224 input image when running on Intel Xeon E5-2620 v2 at 2.10 GHz [34]. The large amount of computational resources and memory costs limits its applications on resource-constrained devices. Therefore, in order to implementing CNNs in such embedded devices, the CNNs have to optimised to be small and efficient. In this PhD work, both the computational cost and memory usage of CNNs are considered to be optimised while minimising the loss in classification accuracy.

Classification Accuracy

When designing a CNN, the first objective would be *classification accuracy* of the model. After training a CNN on a specific training dataset, the model are then tested on the test dataset for evaluating the model's accuracy. The accuracy is defined as:

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} \times 100\% \tag{2.3}$$

In some CNN designs, both *Top-1* and *Top-5* accuracy are reported as the model's accuracy [5, 7, 30]. The *Top-1* accuracy is the conventional accuracy which defined as model's highest probability prediction must be exactly matched with target label. The *Top-5* accuracy means that if any of the top 5 highest probability predictions matches the target label. In this PhD work, it considers the exact accuracy of an optimised CNN. Therefore, the *Top-1* accuracy is considered as the *classification accuracy.*

2.1.3 Computational Cost and Memory requirement of CNNs

When running CNNs on a specific hardware platform, the overall performance is not only about the models' accuracy, the computational resource cost and memory

requirement are also factors to be considered. In this section, the computational resources cost for processing a CNN is discussed.

The relationship between the top-1 classification accuracy, the number of operations required to process the input data for the respective CNNs, and the parameter size of recently constructed CNNs on ImageNet is shown in Fig. 2.7a [13]. In this case, the number of operations is calculated by the number of Multiply-accumulate (MAC) operations for processing a CNN. Since they consider the multiplication and addition of a MAC as two separate operations, the overall number of operations for a convolutional layer in a conventional CNN is defined as:

$$NO_{conv} = 2 \times O_i^2 \times O_n \times K_i^2 \times K_n \tag{2.4}$$

where O_i is the size of output feature map in convolutional layer i, e.g. for a 100×100 output feature map, the O_i^2 is $100 \times 100 = 10^4$. The O_n represents the number of output feature maps in current convolutional layer. K_i represents the size of the convolution kernel and K_n is the number of convolution kernels in convolutional layer i. For example, a VGGNet has 64 3×3 kernels in the first convolutional layer, the K_i^2 will be $3 \times 3 = 9$ and K_n is equal to 64.

For the fully-connected layer i with N neurons, the number of operations for a fully-connected layer is defined as:

$$NO_{fc} = 2 \times N_{i-1} \times N_i \tag{2.5}$$

where the N_{i-1} indicates the number of neurons in previous layer and N_i indicates the number of neurons in current layer.

Fig. 2.7b demonstrates the trend between the number of operations and the forward time of single input images for corresponding CNNs on ImageNet [13]. It can be seen from the Fig. 2.7b, the forward time for processing a single input image on ImageNet is nearly proportional to the number of operations. Therefore, based on the relationship between forward time per image and the number of operations of the model, if the number of operations is reduced, the processing speed of the network will also be reduced.

In this PhD work, the number of operations for a specific model is counted by the number of multiplications. Unlike A. Canziani et al [13], where the multiplication and addition are considered as two separate operations, in this work, the number of MAC operations is counted as a single operation, which is represented by number of multiplications. In this case, two equations for counting number of multiplications in a convolutional layer and a fully-connected layer are:

$$Operation_{conv} = O_i^2 \times O_n \times K_i^2 \times K_n \tag{2.6}$$

$$Operation_{fc} = N_{i-1} \times N_i \tag{2.7}$$

In (2.6), O_i is the size of the output feature maps in convolutional layer i. O_n is the number of output feature maps in current convolutional layer. K_i and K_n represent the size and the number of kernels in convolutional layer i. In (2.7), N_{i-1} is the number neurons in previous layer and N_i is the number of neurons in current fully-connected layer.

In general, the total amount memory for storing weights and biases is counted as:

$$M_{size} = N_{weight} \times B_{weight} + N_{bias} \times B_{bias} \tag{2.8}$$

where N_{weight} is the number of weights, and N_{bias} is the number of biases for a given model. B_{weight} and B_{bias} represent the bit-width of weights and biases, respectively.

2.2 Evolutionary Optimisation

2.2.1 Introduction to Evolutionary Algorithms

In order to generate a set of possible solutions for the problem provided, evolutionary algorithms (EAs) employ several core concepts of biological evolution, such as mutation, crossover and natural selection. Each of the solutions will be ranked by a function which shows the performance of the solution for the targeted objective. In this section, an overview of recent research in evolutionary algorithms is provided.

2.2.2 Basic Operation in Evolutionary Algorithms

For implementing an evolutionary algorithm, there are some basic operation needs to be considered.

Representation: In evolutionary algorithms, specific data structure needs to be considered, which is used for representing solutions and variation operators. This step is also called genetic encoding. Depending on the specific problem, different data representations are needed to be considered with a feasible range. In this process, the target problem is described in a set of variables. This set of variables is defined as *gene*.

Finally, an individual for the target problem can be represented by the combination of specific genes.

Initialisation: For running an EA, the initial population needs to be generated first. The initial population is usually initialised either randomly, or with some specific configurations. The size of the population is defined as the number of individuals in the population.

Variation: Since the EA initialises the population or takes individuals which is survived from the previous population, variation is then placed to modify the genes in each individual. There are two different types of genetic operator used for varying the genes, which are mutation and crossover.

The Fig. 2.8 shows an example of how the mutation operation works with binary representation of genes. Based on mutation rate, there are some genes selected from the survived individuals. Those genes are then randomly varied to generate new genes which are within the range of data representation strategy. For instance, the mutation operation using binary representation is demonstrated in Fig. 2.8 where two genes are randomly selected from the parent individual to build the offspring. In this case, the '1' in the parent string is changed to '0' and the '0' in the parent string is changed to '1' to generate the offspring.

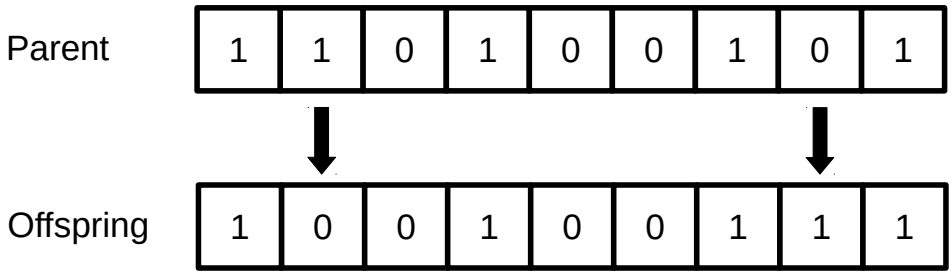


Figure 2.8 An example of mutation operation.

Crossover operation is another type of genetic operation in EAs. The crossover means that two chromosomes exchange part of their genes to generate two new offspring. Different from the mutation operation, the crossover operation does not generate new genes but arranging the existing genes in new combinations. The probability that two chromosome can recombine some of their genes are defined by the crossover rate, which is the frequency of crossover for individuals in a population. In this case, only part of the individuals in offspring population are reproduced by the crossover operations, others in the offspring population are replicated from parent population. Fig. 2.9 demonstrates how the crossover operation is performed on two crossover point. Recent research shows that the mutation operation plays a more important role than the crossover for implementing EAs in some optimisation problem, in some cases, EAs even can achieve better convergence when applying mutation operations only [35].

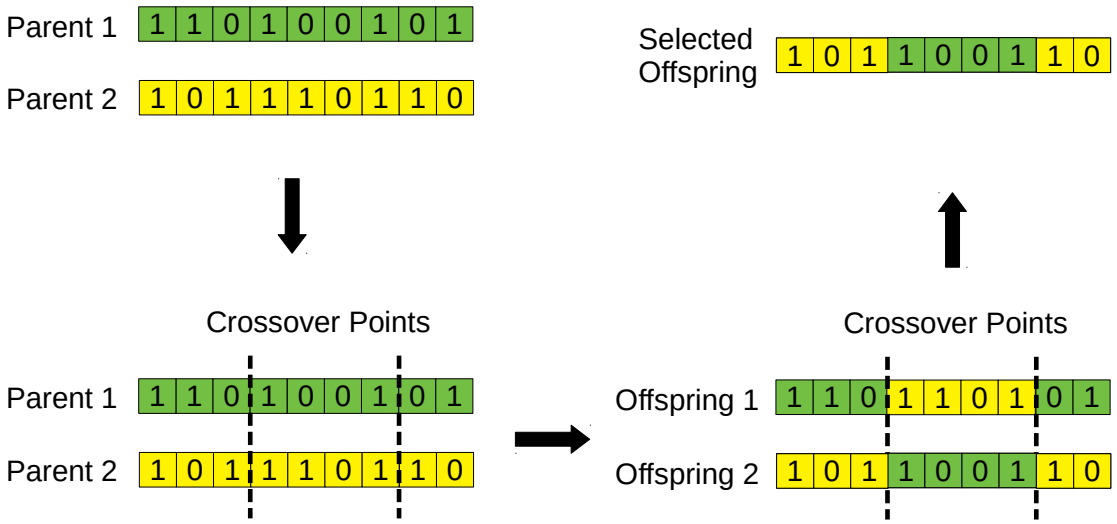


Figure 2.9 An example of crossover operation.

Evaluation: The evaluation process means that measure the performance of each individual in current population. In order to evaluate the performance of the individual,

fitness function needs to be defined as the objective of EA. Then, a numerical scores will be given to the individual that represents its performance.

Termination: Termination means when the EA should stop. This step depends how user wants to terminate the process, e.g. running for a number of iterations or the results cannot improve over a certain of generations.

Selection: After evaluating each individual and if the EA has not been terminated, the selection process will rank the individual based on its fitness score which is evaluated from the evaluation process. The individual with higher fitness score will be passed to the next generation and the individual with lower fitness score will be removed. Elitism strategy is frequently employed since it can guarantee the the individual with the most desirable fitness score are retained unaltered and passed down to the following generation [36].

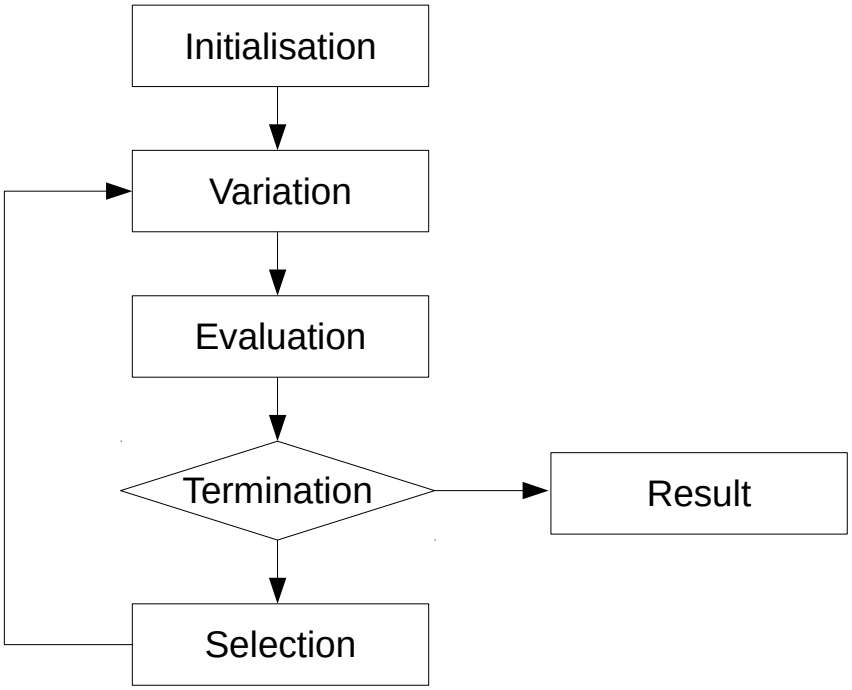


Figure 2.10 The process of evolutionary algorithms

The general implementation of EAs is demonstrated in Fig. 2.10. In the following sections, a range of EAs has been reviewed and analysed.

2.2.3 Evolutionary Strategies

Evolutionary strategies are population-based optimisation method where genetic operators are placed to improve the fitness score of individuals over many generations. In comparison to other optimisation methods, such as stochastic gradient descent for neural networks training which has to build complex optimisation function, the evolutionary strategy is a black box algorithm that influences the results through intervention and gradually selects the optimal point through iterations. It is a kind of black box algorithm that does not consider the intermediate complex functions. The optimisation goal can be achieved directly by simply defining the reward.

The key distinctions between various evolutionary strategy implementations are how the parent and offspring populations are handled and what kinds of genetic operators are utilised to derive the offspring populations. There are some algorithms that only involves mutation operations and others using both crossover and mutation as genetic operators to generate the offspring [37–39]. The simplest evolutionary strategy is called (1+1) Evolutionary Strategy ((1+1)-ES). As the name suggests, the (1+1)-ES is the optimisation algorithm that one offspring is produced by Gaussian mutation [40] by one parent [41]. In this method, there is only one parent individual and only producing one offspring at a time, and then, keeps the individual with better fitness as the parent to the next generation. Another two methodologies of evolutionary strategies are (μ, λ) Evolutionary Strategy ((μ, λ)-ES) and $(\mu + \lambda)$ Evolutionary Strategy ($(\mu + \lambda)$-ES), where μ is the number of individuals that used to generate offspring, λ is the number of individuals in offspring population [39]. The main difference between (μ, λ)-ES and $(\mu + \lambda)$-ES is that, in (μ, λ)-ES, the offspring population is evaluated and select μ individuals as the parent to generate the offspring for next iteration. In $(\mu + \lambda)$-ES

both the parent and offspring population are evaluated together, then, the number of μ individual are selected to generate new offspring for next generation by comparing the best fitness in both parent and offspring.

A self-adaptive evolutionary strategy has been proposed to deal with continuous optimisation problems, called Covarience Matrix Adaptation Evolution Strategy (CMA-ES) [42]. In CMA-ES, for the multi-dimensional vector, the covariance information is put into the matrix, and the random points are generated based on the multi-dimensional Gaussian distribution of the covariance matrix. In addition, the historical step is maintained to adjust the parameters of the multi-dimensional Gaussian distribution smoothly. In this way, more samples are collected in potential areas while ignoring less important factors and it can get results more quicker than (μ, λ)-ES and $(\mu + \lambda)$-ES.

2.2.4 Multi-objective Evolutionary Algorithms

In CNN design, many scenarios will come up with trading-off between multi-objective. For instance, researchers are usually required to design a CNN with high classification accuracy, low memory costs, less power consumption and fast processing speed. Hence, only focusing on improving CNNs classification accuracy has been no longer meeting the design requirements and goals. In addition, a good design with multiple greater performance is necessary for today's AI applications.

Problems with multiple objectives arise naturally in any real-world issues and solutions seeking is challenging [17]. Dealing with multi-objective optimisation problems is impossible for the optimisation of a particular solution only with respect to one single target leading to unacceptable results in terms of remaining objectives [16]. To define a multi-objective problem with n decision variable vectors and m objectives , it can be described as (2.9) [17],

$$F(x) = [f_1(x), f_2(x), ..., f_m(x)]^T, x = [x_1, x_2, ..., x_n]^T \in X \subset \mathbb{R}^n \tag{2.9}$$

where x is the n-dimensional decision vector and X is the n-dimensional searching space. The objective $F(x) \subset \mathbb{R}^m$ is the m-dimensional objective space. The x defines m functions mapping X to $F(x)$.

It eventually does need to iterate a set of compromised solutions of trade-offs in regarding to all proposed objectives from which users could select proper solution. The trade-offs between multiple objective functions, $f_1(x), f_2(x), ..., f_m(x)$ is normally adopted notion of *Pareto optimal* [43].

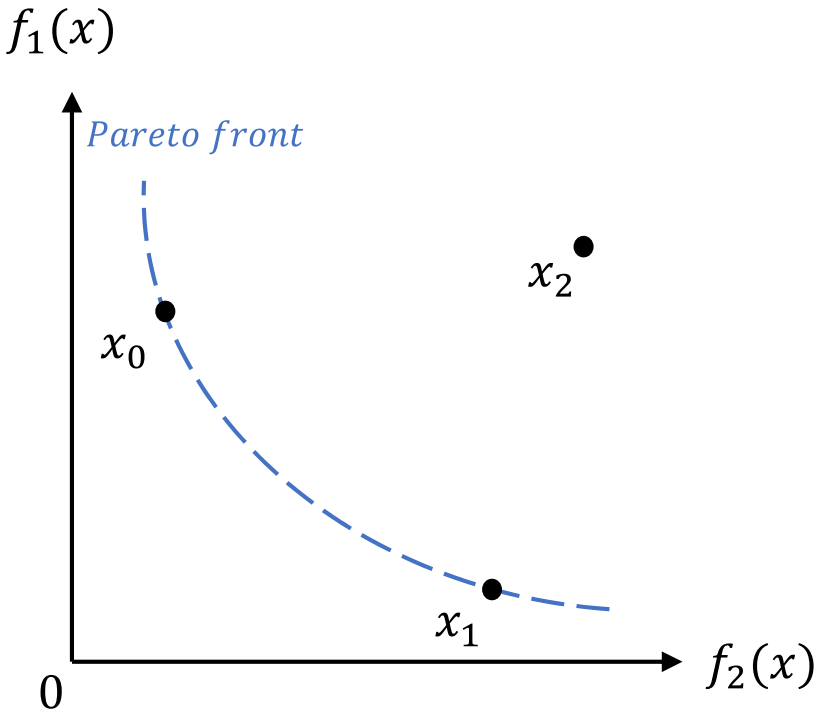


Figure 2.11 The *Pareto optimal* between two objective. Both solution x_0 and x_1 are lying on the *Pareto front*, which means these two solutions are not dominated by any other solutions in the searching space X. The solution x_2 is not in the *Pareto front*, since it is dominated by x_0 and x_1.

The Fig 2.11 demonstrates an example of the *Pareto optimal* for a two objective minimisation problem. The solution x_0 and x_1 are identified on the *Pareto front* and both solutions outperform the x_2, i.e. the trade-offs between objectives. In this case, if solutions are not *dominated* by any other solutions, they are defined as Pareto optimal or trade-offs. The *dominate* is defined in (2.10), where it aims to find the possible solutions that can satisfy with one objective without making others worst.

$$f_i(x_0) \leq f_i(x_2), i = 1, 2, ..., m \tag{2.10}$$

As the results, multi-objective evolutionary algorithms (MOEAs) are one of today's most powerful techniques for solving multi-objective optimisation problems [44, 45]. Vector Evaluated Genetic Algorithm (VEGA) is the first MOEA which propose by D. Schaffer [46]. Later on, many powerful MOEAs have been proposed, such as Multi-objective Genetic Algorithm (MOGA) [47], Non-dominated Sorting Genetic Algorithm (NSGA) [48] and Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [18] which are widely used in many applications.

In VEGA, there is a number of sub-populations that are selected from the whole population in every generation, then, each of the sub-population are only assigned one optimisation objective [46]. For example, if the user aims to solve a problem with n objectives and the number of individuals in each sub-population is p, hence, the population size of VEGA will be $n \times p$. In this case, the VEGA is working as running multiple single-objective evolutionary algorithms simultaneously. However, this approach gets solutions that excel in one purpose without considering the performance for other objectives, which is a dilemma that prohibits finding compromise solutions with respect to tasks which need to be completed.

MOGA proposed non-dominated ranking and selection to make the optimisation result can converge to all objectives at the same time [47]. In this algorithm, rank 1 would be

assigned to non-dominated individuals, which will be assigned the highest fitness and other ranks would be assigned to dominated individuals based on their neighbouring population density. A similar approach was taken by the NSGA in order to maintain the population diversity [48]. The niche radius, σ_{share}, is used to measure how crowded each individual's region is. In this case, specifying the niche radius appropriately is critically important to the performance of MOGA, which bring another challenge to the such algorithm.

The NSGA-II was developed by K. Deb et al in 2002 as an improved version of the original NSGA [18]. In the NSGA-II, through the implementation of the non-dominated sorting approach and the diversity preservation strategy, the ability to select solutions without elitism can be retained and a wide spread of solutions can be explored. In this algorithm, there are two features for each solution: 1) how many solutions are dominated by the target solution, 2) how many solutions dominate the target solution. In this case, if the solution is not dominated by other solutions, this solution will be placed into first rank. There are multiple levels of ranking depending on the domination count of the solution. And also, the niche radius, σ_{share}, is no longer used in NSGA-II, instead, the NSGA-II introduced crowding distance measurement to spread the diversity of solutions. For example, if two individuals are in the same rank, the individual with less crowding distance are considered as better solution [44]. These two approaches perform the NSGA-II can not only converge to all objectives simultaneously, but also can explore the solution space widely without additional settings of niche radius or other parameters.

2.3 Evolutionary Optimisation in Neural Networks

In recent years, the recognition accuracy of neural networks increases, the model size and parameter size are also rising dramatically [13]. As the results, designing and

optimising neural networks by evolutionary algorithms have also drawn attentions to researchers [49].

Since designing CNNs requires to define different functional layers and topology, the EAs are therefore required to configure to represent network topologies. In general, there are three main categories used for evolving CNNs by EAs, which are layer-based evolution [50–58], module-based evolution [59–67] and topology-based evolution [19, 68, 69]. In this section, recent research approaches have been reviewed from these three perspectives.

Functional layers are the basic unit for designing a neural work, such as convolutional layers, pooling layers and fully-connected layers. For a well-designed CNN, it usually contains multiple functional layers and each layer has different parameter settings. For example, five parameters need to be set up for implementing a single convolutional layer, which are convolution kernel height and width, number of kernels in the convolutional layer (convolutional layer depth), stride and padding methodology. For a pooling layer, the design is required to define what kinds of pooling strategies are necessary, such as max pooling or average pooling, and also, defining the size and stride are also essential. Therefore, in layer-based evolution, there are so much parameters encoded in the search space which results extremely large search space. This makes EA take a lot of time to converge to a solution through layer-based evolution. Nevertheless, a large search space means that there is more possibility to find the global optimal for designing CNNs on target datasets [70].

In the past ten years, different convolutional modules have been introduced for designing CNNs and each convolutional module normally has multiple types of function layers [7, 5, 30, 71, 72]. On the one hand, those convolutional modules contain typical connections and size between functional layers which results in improvement in CNNs' classification accuracy without vanishing gradient and exploding gradient. On the other hand, designing CNNs by using these function layers can make the model easily extend to

deeper connections, and thus, provide high classification accuracy [5, 30]. These special connections of different convolutional module cannot be found by using layer-based evolution. Therefore, module-based evolution becomes necessary for optimising these types of CNNs. Moreover, the connections inside those convolutional modules are fixed, in order to guarantee the functionality and performance of the convolutional modules. Therefore, less parameters need to be determined when applying module-based evolution, compared with layer-based evolution. for example, M.Suganuma et al [64] applied EAs to encode the CNN architectures based on ResBlock [30] and customised ConvBlock which leads significant improvement in model's classification accuracy than standard VGGNet [5] and ResNet [30].

Different from layer-based evolution and module-based evolution where the size of layer or modules is parameterised into EAs' representation, the topology-based evolution only consider the connection between different layers or modules. For instance, CARS [68] proposed a topology-based evolutionary optimisation method for maximising the knowledge that is learned from previous generations. The CARS initialises a large neural network, called SuperNet, which contains a large amount of different blocks and cells. Then, a number of subgraphs, i.e. different connection between blocks and cells, are generated from the SuperNet by EA. These subgraphs are trained on target dataset and updating the cells in the SuperNet.

2.4 Summary

In summary, this chapter provides an overview of artificial neural networks, typically for convolutional neural networks which are this thesis focused on, as well as evolutionary algorithms and its applications in optimising CNNs.

From the study and investigation of CNNs, there are two main challenges in processing CNNs. Firstly, it is obvious that high performance CNNs require very deep structure,

as the network goes deeper and wider, it requires more computation resources to process such networks. Secondly, those deep CNNs contain huge amount of parameters which requires a large memory size to storage data. The CNNs bring significant challenge for computational resources and memory size, especially for resource-constrained embedded devices. Therefore, optimising CNNs to reduce the computational cost and model size while remaining high classification accuracy is critical.

Evolutionary algorithms demonstrate a good performance in optimisation problems which can effectively deal with one or more objectives with providing multiple better solutions for decision makers. Past research show that evolving the CNN architectures by EA can achieve higher classification accuracy than human-designed CNNs. It is an opportunity for evolutionary algorithms to optimise the CNNs architectures that balancing the computational costs, memory consumption and model's classification accuracy at the same time.

Chapter 3

Evolutionary Optimisation of Kernel Shapes and Sizes

While state-of-the-art development in Convolutional Neural Networks (CNNs) topology makes CNNs' classification accuracy become increasingly accurate, these networks have became computationally expensive involving billions of arithmetic operations and parameters. 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 solutions. In this chapter, a multi-objective optimisation approach is proposed to trade-off between the amount of computation required and network accuracy, by using a Multi-Objective Evolutionary Algorithm (MOEA). The number of convolution kernels and the sizes of these kernels, are directly proportional to computational resource requirements of CNNs. Therefore, this section considers optimising the computational resource consumption by reducing the sizes of kernels in convolutional layers. Additionally, the use of unconventional kernel shapes is investigated, and the results show these clearly outperforming the commonly-used square convolution kernels.

3.1 Overview

In order to achieve a high classification accuracy, state-of-the-art CNN architectures have grown extremely complex. For example, AlexNet [6] requires billions of multiply-accumulation (MAC) operations to process a single image. In general, convolutional layers are the most computationally expensive, where the resource consumption in each layer is proportional to the number and sizes of the convolution kernels. For instance, a 5×5 convolution kernel requires more than twice the number of MAC processes than a 3×3 kernel. Therefore, to reduce resource consumption in convolutional layers, this work aims to reduce the number of MAC processes in these layers by

using unconventional (non-square) kernel shapes as well as reducing the number of kernels in each convolutional layer with minimum loss of network accuracy. The conventional approach to designing the convolutional layer in CNNs is using a set of square convolution kernels which extract the image features. However, the conventional kernels are computationally costly where a large number of multiplications is needed to calculate each feature map. For example, a 5×5 convolution kernel needs 25 MAC operations to compute, but a 1×5 convolution kernel only requires five MAC process. Therefore, this work aims to find out whether the unconventional convolution kernels can be to replace the square convolution kernels in CNN designs. In [73] the idea of separable convolutions is explored, where a set of smaller, one-dimensional (1-D) convolution kernels is designed to replace the conventional two-dimension (2-D) ones. For example, a specific $n \times n$ matrix can be replaced by the product of a $n \times 1$ matrix times a $1 \times n$ matrix, instead of calculating one convolution with nine MAC processes, the reformulation calculates two convolutions with three MAC processes each, totally six MAC processes, to achieve the same performance for specific tasks. The caveat is that this separation only works within certain constraints, i.e. only a subset of matrices with specific properties can be replaced. Our proposed design contains a number of differently-shaped kernels, including 1-D kernels, 2-D rectangle kernels and 2-D square kernels. Each of these kernels can have a different shape and size which require different numbers of MAC processes to extract the features from the input images.

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 [15]. 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 topology 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 [19–21]. Fast Non-dominated

Sorting Genetic Algorithm (NSGA-II) [18] was proposed in 2002. The non-dominate sorting approach of NSGA-II can make the target problem converge to two or more objectives simultaneously, and the crowding distance measurement will spread the diversity of solutions which can provide a large trade-off space between objectives. Those features of NSGA-II are naturally compatible with the design requirements of efficient CNNs, which consider both computational cost and the model's classification accuracy at the same time.

In order to both minimise the resource consumption and retain the network accuracy while processing the CNNs on the target hardware platform, NSGA-II [18] is applied 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 the network width using combinations of various unconventional shapes and sizes of kernels, while keeping the depth constant.

This chapter is organised as follows: Section 3.2 gives an overview of the current approaches of optimising feed-forward neural network architecture, as well as recent approaches for applying unconventional convolutions. Section 3.3 describes the design methodology that implements NSGA-II to explore the design space of CNN architecture and explains its features. Section 3.4 shows several test experiments with varying benchmark datasets and provides a proof-of-concept of the performance of the proposed method. Finally, Section 3.5 summaries the chapter.

3.2 Related Works

3.2.1 Optimisation of Network Architecture

Evolutionary Algorithms (EAs) are widely used in optimisation problems with complex fitness landscapes. When optimising artificial neural networks using EAs, the main idea is to evolve the synaptic weights and connections of the network [21, 50, 51, 59, 60, 68, 69], where EAs are placed to adjust the networks' topology by changing the connections between layers or neurons. In contrast, NEAT [74] is a method that uses genetic algorithms (GAs) to change both connection weights and network structure at the same time. 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 [75, 76]. However, due to the large 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 its size. G. Morse and K. Stanley [77] compares 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.

Based on previous approaches, it is reasonable to assume that GAs are 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. E. Real et al [78] 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 grown 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 also dramatically increased.

CoDeepNEAT [79] is a further extension of NEAT [74] 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 node 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 node in the blueprint is replaced with a module chosen randomly from the species to which that node points. The results show that the network designed by CoDeepNEAT can achieve competitive accuracy in image classification problems with faster training speed.

Y. Kim et al [80] 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 [19] 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 population, such as pre-defining the layer functionalities and connectivity, there have also been developed some GA-based fully automatic architecture design methods in recent years, e.g. [81]. Their design methodology contains a building block that acts as 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, M.Suganuma et al [20] proposes using Cartesian Genetic Programming (CGP) [82] 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. 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. Those previous investigations suggest that applying MOEA for optimising CNNs may potentially provide good trade-off between the computational resource cost and model's classification accuracy.

3.2.2 Unconventional Convolutions

Conventional CNN designs use square kernels to detect the image features. This design method brings significant challenges for computational systems, because the number of arithmetic operations increases 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 [73,

83, 84] 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 less computations. Therefore, the 1-D convolution approximation can significantly accelerate the classification speed as well as reducing the number of network parameters. Similarly, J. Jin et al [85] introduce this into the training phase by factorising a conventional three-dimensions (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 capable of processing ImageNet datasets [86]. 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. [87] 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. [88] 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.

3.3 Methodology

The main idea in this work for optimisation of the CNN architecture is to consider how the classification accuracy can be improved (or kept the same) while reducing the computational resources required. Previous investigations suggest that using different shapes of convolution kernels can improve the network performance by detecting multiple-scale features from input images [88, 87]. 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 [18] to automatically discover efficient network architecture by finding the trade-off between the computational complexity and module classification accuracy on the three benchmark datasets, MNIST [8], Fashion-MNIST [89] and CIFAR-10 [9].

3.3.1 Computational Resource Consumption

For state-of-the-art CNN architectures, the computationally most expensive parts are the convolutional layers [13]. 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. 3.1. Mathematically, the formula for calculating a single feature map by convolution operation can be described as (3.1),

$$O_{:,:,oc} = \sum_{i=1}^{IC} I_{:,:,i} \otimes K_{:,:,oc} \tag{3.1}$$

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.

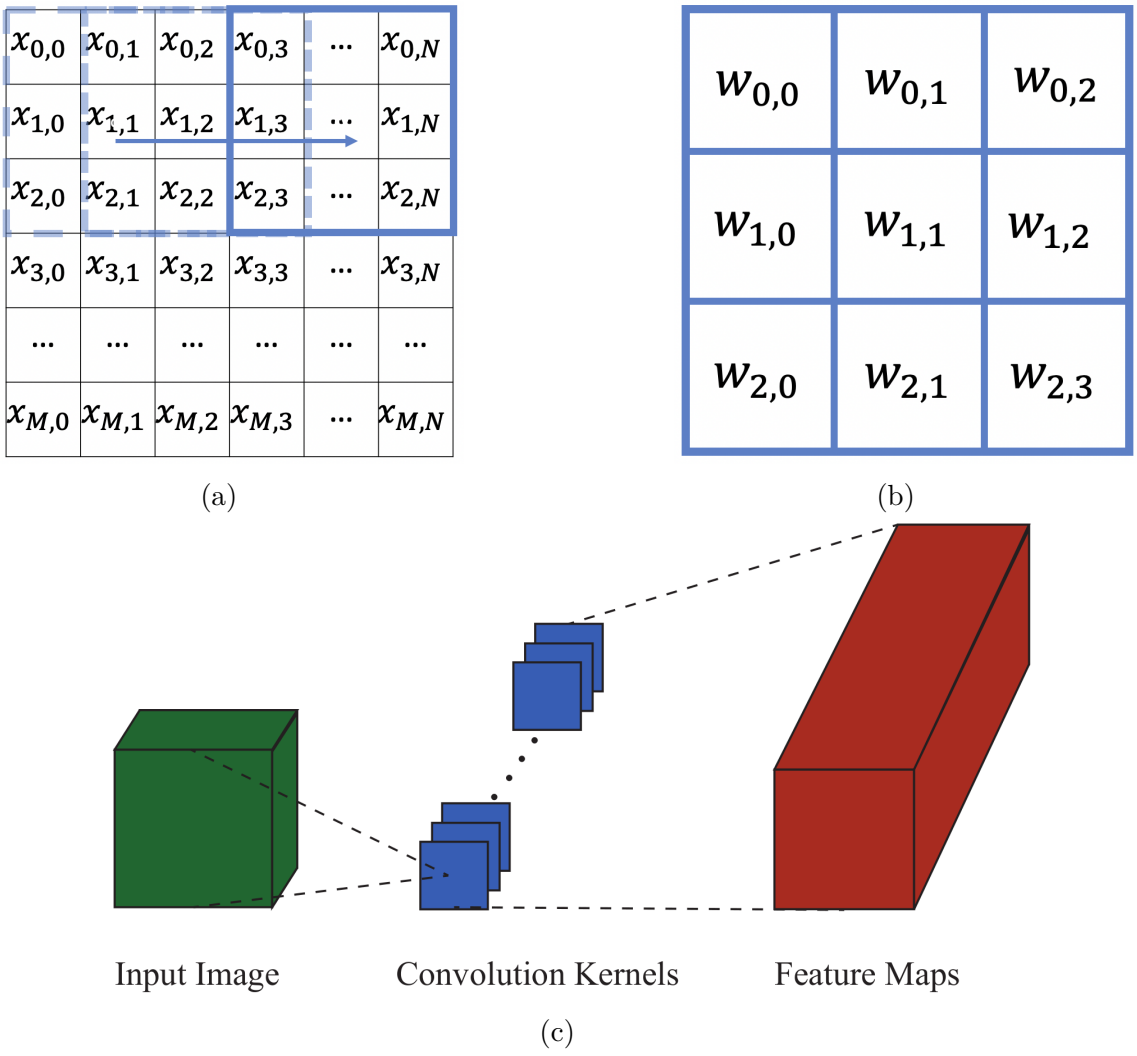


Figure 3.1 (a) An example of sliding a 3×3 convolutional kernel across the input image. (b) A conventional 3×3 square kernel that is used to extract features from the input image. (c) Example of a conventional convolutional layer with multiple square kernels.

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:

$$Operation_{conv} = O_h \times O_w \times O_c \times K_h \times K_w \times K_c \tag{3.2}$$

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 (3.2), the number of operations in the fully-connected layer of a specific CNN can be calculated as:

$$Operation_{fc} = O_h \times O_w \times O_c \times N_i \tag{3.3}$$

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.

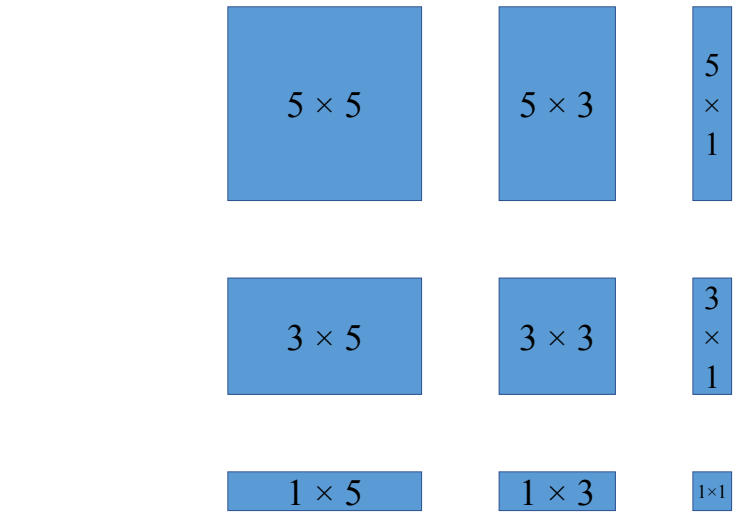
As can be seen from (3.2) and (3.3), the convolutional layer requires significantly more MAC operations than the fully-connected layer. Therefore, reducing the number of MAC operations can significantly reduce the computational resource consumption overall. This is important to allow state-of-the-art 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.

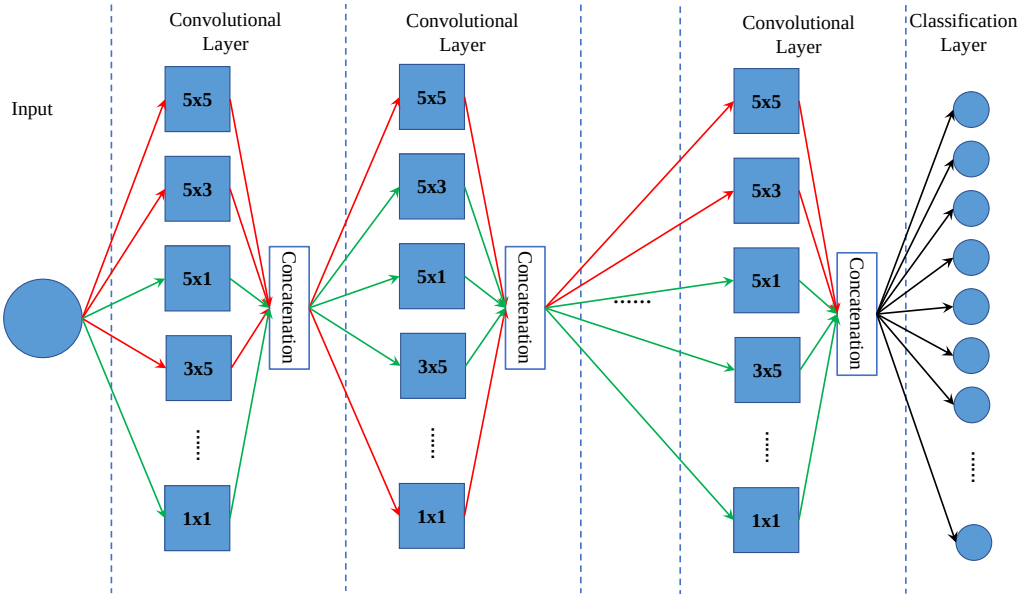
3.3.2 Mixed Unconventional Kernels

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 (3.2). 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 [87], 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. 3.2.



(a)



(b)

Figure 3.2 (a) The set of unconventional kernels. The number of operations required to compute each of the unconventional kernels can be calculated by kernel height × 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 strategy 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. 3.3.

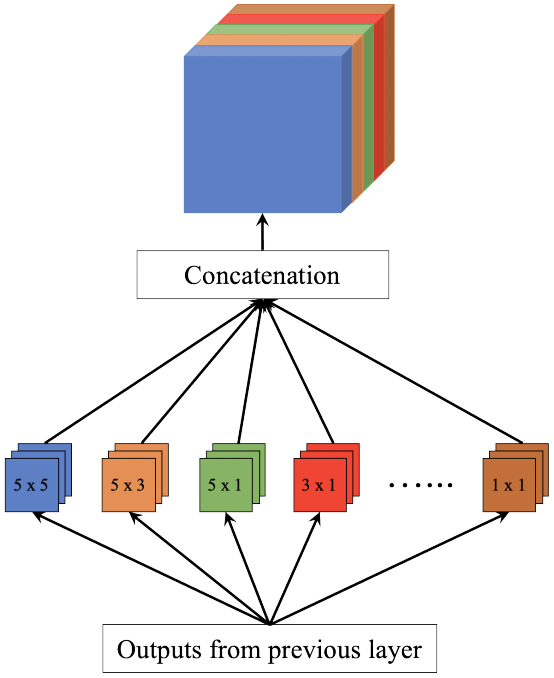


Figure 3.3 Overall design of the convolutional layer which involves multiple sizes of kernels to produce different feature maps. If the strides of each type of kernel are the same, using the padding method, the output from each set of kernels will be the same as other kernels. Therefore, the final output from this layer can be concatenated into a single tensor with no additional computation operation required.

3.3.3 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 [18]. 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. 3.4.

In the optimisation loop, each of the possible unconventional kernels is identified with its kernel ID that represents a specific kernel. The kernels from all layers of the CNN are encoded in the genotype. Table. 3.1 illustrates the genetic representation for the MOEA. In the optimisation loop, convolution kernels are encoded as integers from 0 to 8, where each integer represents a size of convolution kernel. As we are focusing on optimising the 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. The mutation process is presented in Algorithm 1. Crossover operation is not used in this case.

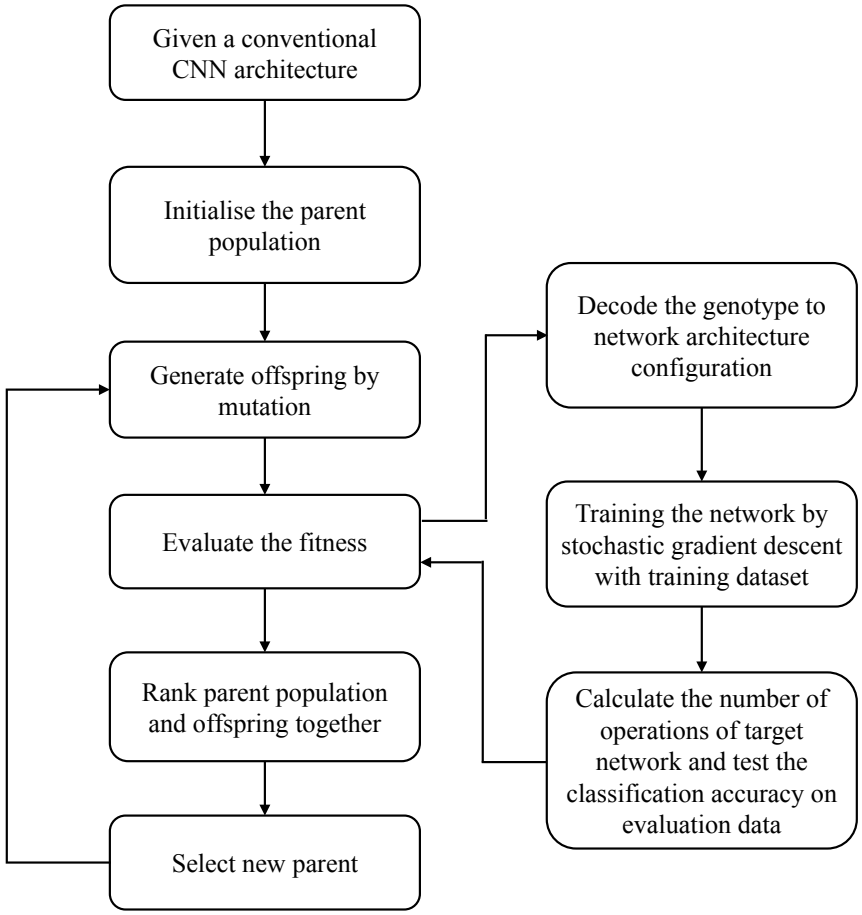


Figure 3.4 Overview of the multi-objective optimisation loop. The method selects a set of unconventional kernel shapes replacing the original (conventionally used) square kernels in a given CNN architecture. Each individual is trained on a training data set. Then, the fitness is calculated and assigned 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.

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 error which is defined as **1 - Top-1 accuracy** of trained model on the test dataset. Another fitness is

Table 3.1 Genetic representation for convolution kernels

Representation	Convolution Kernel	Number of Multiplications
0	1 x 1	1
1	1 x 3	3
2	1 x 5	5
3	3 x 1	3
4	3 x 3	9
5	3 x 5	15
6	5 x 1	5
7	5 x 3	15
8	5 x 5	25

Algorithm 1 Mutation process

Procedure: Mutation (P, ρ). ▷ Population (P) has N individuals and M genes in each individual. The mutation probability is ρ.

```
Offspring population (O) ← P
for i ← 1 to N do
    for j ← 1 to N do
        if random(0, 1) < ρ then
            O_{i,j} ← randomInt(0, 8)
        end if
    end for
end for
```

the summation of MAC operations required to compute all of the convolutional layers in the network, which is calculated by (3.2). Therefore, the fitness function will be:

$$f(x) = \min\ [classification\ error,\ number\ of\ multiplications] \tag{3.4}$$

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, which calculates the distance of each individual to others. 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 [44]. Following this, half of the individuals which have higher rank will be selected as the parent population for the next generation. The optimisation loop is demonstrated in Algorithm 2.

Algorithm 2 NSGA-II for optimising kernel shapes and sizes

Procedure: NSGA-II (M, N, $f(x)$). ▷ evolving N individuals for M generation with fitness function of $f(x)$, $\forall x \in X$, where X is the set of convolution kernels.

Initialise parent population $P_1 = [x_1,\ x_2,\ ...,\ x_N]$
Offspring population $(O_1) \leftarrow$ Mutation (P_1)
for $i \leftarrow 1$ *to* M **do**
  $C_i \leftarrow O_i \cup P_i$ in size $2N$
  $R_i \leftarrow f(C_i)$ ▷ evaluating each individual in C_i with fitness function $f(x)$
  $F \leftarrow$ Non-Dominated-Sorting(R_i)
  $P_{i+1} \leftarrow \emptyset$
  $j \leftarrow 1$
  while $|P_{i+1}| + |F_j| \leq N$ **do**
    Crowding-Distance-Calculation(F_j)
    $P_{i+1} \leftarrow P_{i+1} \cup F_j$
    $j \leftarrow j + 1$
  end while
  $F_j \leftarrow$ Descend-Sort(F_j)
  $P_{i+1} \leftarrow P_{i+1} \cup F_j[1 : (N - |P_{i+1}|)]$ ▷ individual with less crowding distance from the first to the $(N - |P_{i+1}|)th$ of F_j to fill P_{i+1}.
  $O_{i+1} \leftarrow$ Mutation(P_{i+1})
end for

3.4 Experimental Results and Analysis

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. The aim in this chapter 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. Therefore, a small CNN 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.

3.4.1 Experimental Settings

The benchmark CNN architecture is built based on the LeNet-5 architecture [27]. The original LeNet-5 consists of three convolutional layers and two average-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. 3.5, which is used as the benchmark topology to test the proposed method.

The optimisation method is applied to three different datasets, MNIST [8], Fashion-MNIST [89] and CIFAR-10 [9]. 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 grey-scale 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-10 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

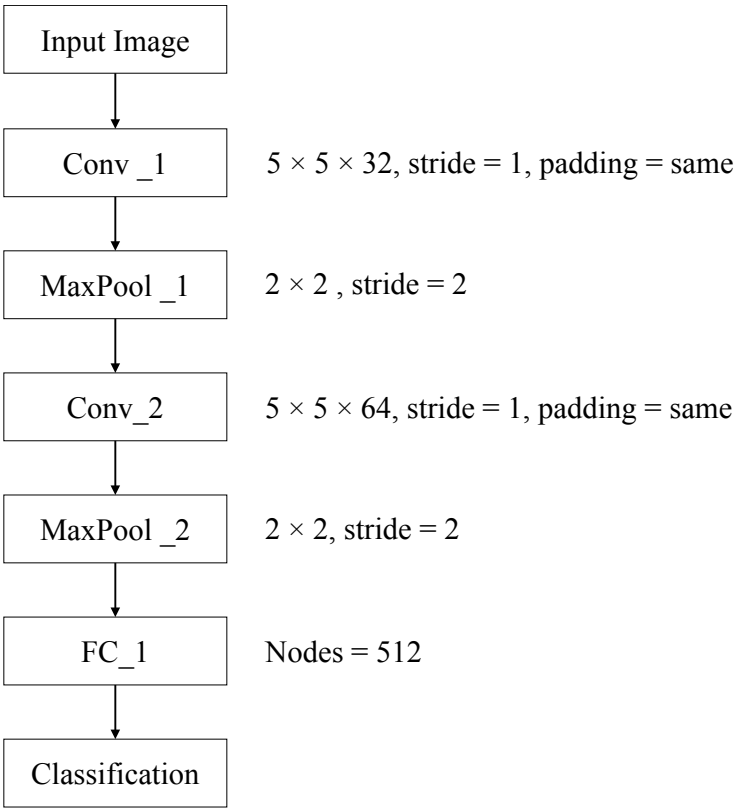


Figure 3.5 The benchmark CNN used to test our optimisation method. The benchmark CNN has four layers that is built based on the LeNet-5 architecture. The network involves two 2-D convolutional layers, which contain 32 and 64 kernels respectively. All of the kernels have dimensions of 5×5 and the stride 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.

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 stochastic Gradient Decent (SGD) method and use the Adam optimiser [90] 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, the mutation rate is 0.1.

3.4.2 Experimental Results

In order to evaluate the performance of the proposed method, it has been tested on three different benchmarks, MNIST, Fashion-MNIST and CIFAR-10 using two optimisation objectives, the total number of multiplications in convolutional layers, and the model classification accuracy. All experiments are implemented on Intel Xeon 6138 20-core 2.0 GHz CPU with single NVIDIA Tesla V100 32GB SXM2 GPU. The approximate runtime of each experiment is reported on Table 3.2.

Table 3.2 Approximate runtime of the proposed method on each dataset.

Model	Dataset	Runtime
LeNet-5	MNIST	$\approx$ 5 days
	Fashion-MNIST	$\approx$ 5 days
	CIFAR-10	$\approx$ 9 days

Results for MNIST and Fashion-MNIST

For evaluating the proposed method on MNIST and Fashion-MNIST datasets, the input layer of the benchmark is configured as $28 \times 28 \times 1$ input neurons. Then, the benchmark CNN is trained for 100 epochs by using the Adam optimiser [90] with an initial learning rate of 0.001, and the learning rate is reduced by factor of 10 at 30th epoch for both training MNIST and Fashion-MNIST. After the training process is completed, the model accuracy for both datasets is evaluated on the test sets. For these two datasets, the benchmark CNN achieves a classification accuracy of 98.92% on MNIST and 92.12% on Fashion-MNIST. Both benchmark networks require a total of 10,662,400 multiplications in their convolutional layers.

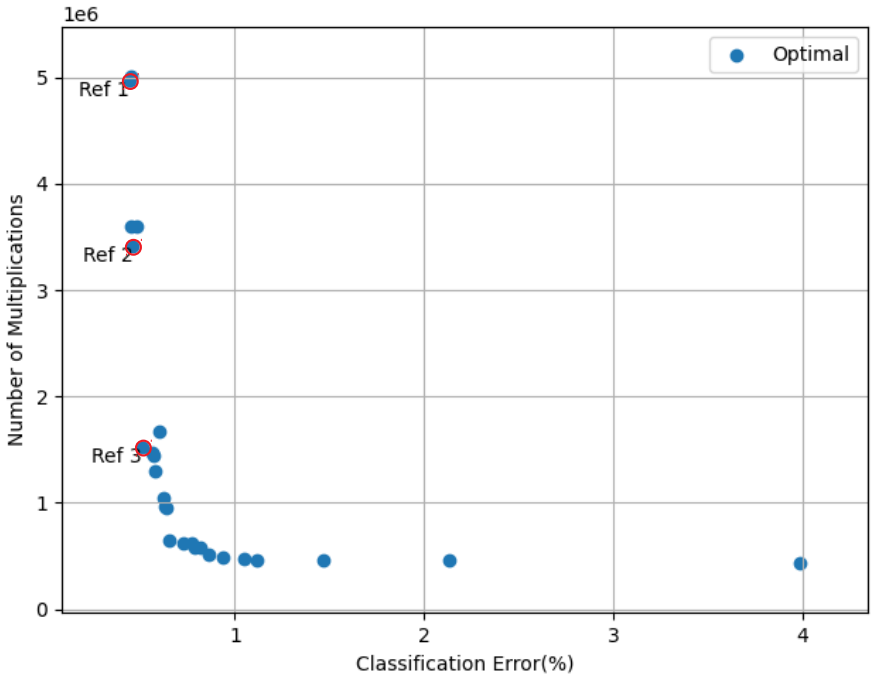
For the experimental results, there are three reference solutions that have been selected from each set of solutions. The first reference point, Ref 1, has the highest accuracy. The second reference point is one which requires significantly less computational resources while still featuring high accuracy. The third reference point is the trade-off solution closest to the origin between number of multiplications and classification accuracy, i.e. the "best" trade-off between the two objectives. The optimisation results after running 100 generations are shown in Fig 3.6, and the comparison between the benchmark network and reference point is shown in Table 3.3.

Table 3.3 Comparison between the benchmark network, i.e. LeNet architecture shows in Fig. 3.5, and solutions found by the proposed method on MNIST and Fashion-MNIST datasets. Three reference points are selected from optimised results for each dataset.

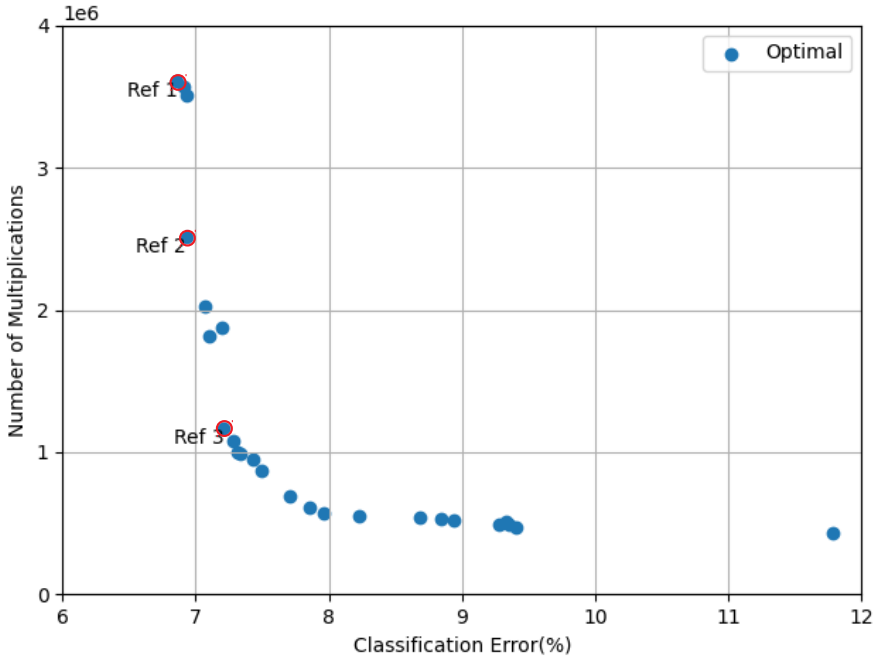
Dataset	Model	Top-1 Acc.	Acc. improve	Reduction in Mults.
MNIST	Benchmark	98.92%	-	-
	Ref 1	99.56%	0.64%	2.15x
	Ref 2	99.54%	0.62%	3.13x
	Ref 3	99.49%	0.57%	7.02x
Fashion-MNIST	Benchmark	92.12%	-	-
	Ref 1	93.14%	1.02%	2.95x
	Ref 2	93.07%	0.95%	4.24x
	Ref 3	92.79%	0.67%	9.15x

Results for CIFAR-10

Then, the proposed method is tested on the CIFAR-10 dataset. 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 the test set. To prevent overfitting, a weight decay of 0.0001 and data augmentation have been used for training the networks. The data augmentation used is based on [91], that is padding 4 pixels on each side and randomly crop a patch of a size of 32×32 from the padded image or its horizontally flipped version. In order to handle the colour



(a)



(b)

Figure 3.6 (a) Optimised architectures by the proposed method after 100 generations on MNIST. (b) Optimised architectures by the proposed method after 100 generations on Fashion-MNIST.

image inputs, the input layer of the benchmark network is presented to the network 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. 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. The optimisation result after running the proposed method for 100 generations is shown in Fig 3.7.

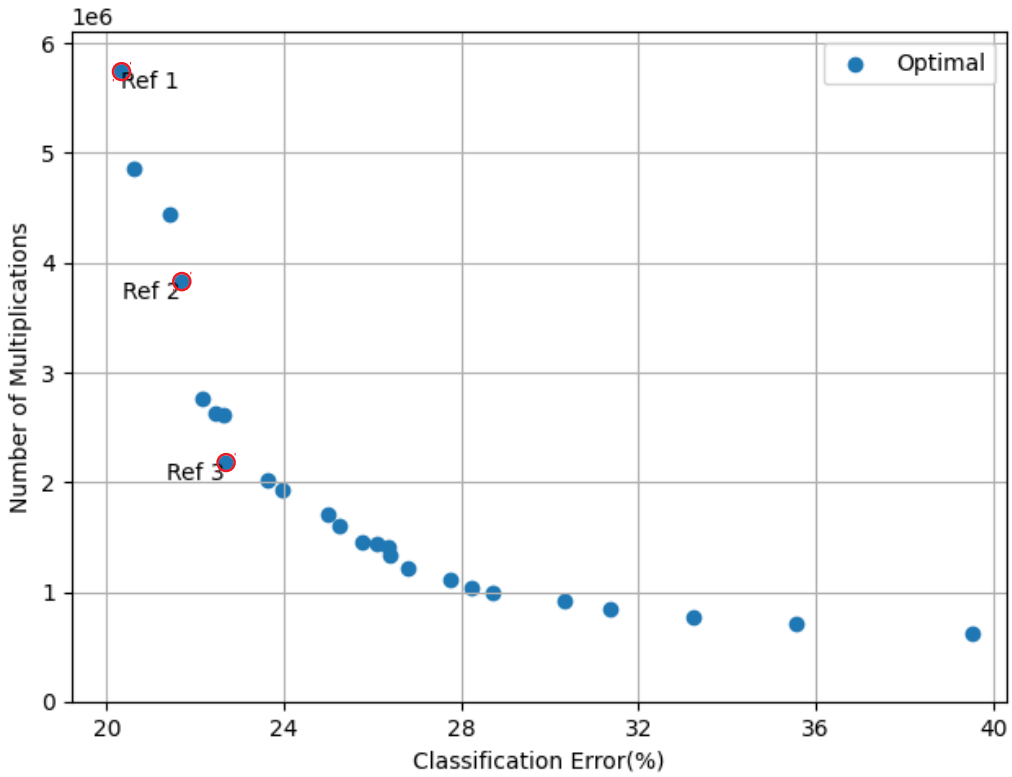


Figure 3.7 Optimised architectures by the proposed method after 100 generations on CIFAR-10.

For further evaluation, the three architectures are then re-trained on the whole training set, i.e. 50,000 images for training, and the classification accuracy is evaluated on the test set which contains 10,000 examples. Each network architecture is re-trained for 100 epochs with weight decay and data augmentation [91]. All of them are trained by using the Adam optimiser [90] with an initial learning rate of 0.001, and the learning rate is reduced by a factor of 10 at the 30th epoch. After re-training and testing, the comparison between the benchmark network and reference point is shown in Table 3.4.

Table 3.4 Comparison between the benchmark network and three reference solutions found by the proposed method on CIFAR-10 dataset after re-training for 100 epochs.

Dataset	Model	Top-1 Acc.	Acc. improve	Reduction in Mults.
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

3.4.3 Convolution Kernels Distribution

From the experimental results, it can be seen that the proposed method shows significant reduction in computational costs for processing CNN. In this section, the details of convolution kernels of optimised CNNs for each dataset has been shown in here.

There are three reference network architectures picked from the optimisation results from each dataset. The first reference point, “Ref 1”, is defined by the optimised result with the highest accuracy. The second reference point, “Ref 2” has a slightly lower classification accuracy than the first reference point but saves more computation resources. The third reference point, “Ref 3”, is the optimised architecture with closest-to-origin trade-off between number of multiplications and classification accuracy.

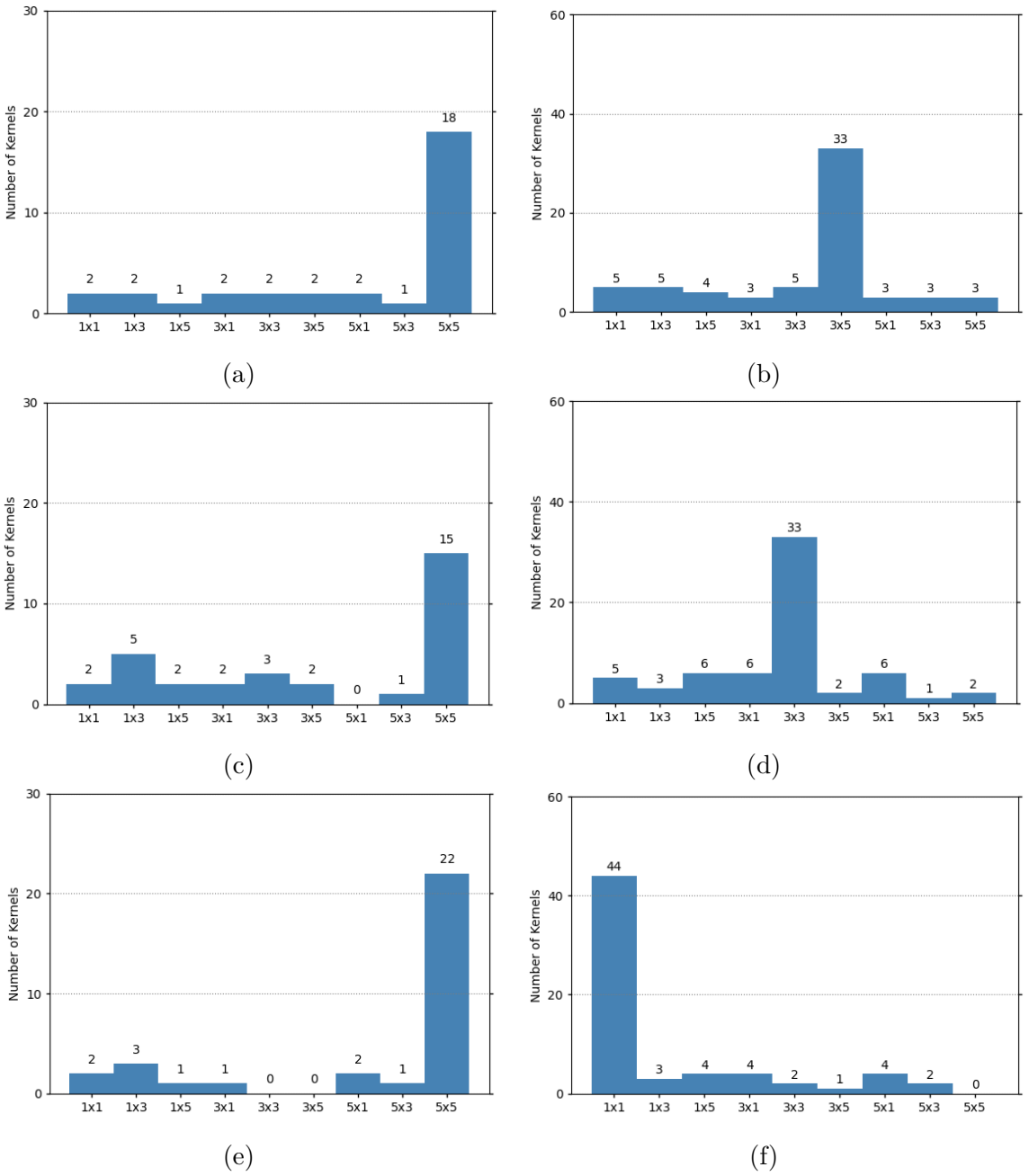


Figure 3.8 (a) Kernel distribution of the first convolutional layer of Ref 1 on MNIST dataset. (b) Kernel dist. of the second conv. layer of Ref 1 on MNIST dataset. (c) Kernel dist. of the first conv. layer of Ref 2 on MNIST dataset. (d) Kernel dist. of the second conv. layer of Ref 2 on MNIST dataset. (e) Kernel dist. of the first conv. layer of Ref 3 on MNIST dataset. (f) Kernel dist. of the second conv. layer of Ref 3 on MNIST dataset.

Fig 3.8 shows details of how the convolutional layers of the benchmark architecture on the MNIST dataset have been built by the proposed method. It can be seen that all these three reference points mainly contain 5×5 kernels in the first convolutional layer. However, the second convolutional layers of the three reference architecture are different. As is shown in Fig 3.8, the kernels in the second convolutional layer of "Ref 3" are mainly of size 1×1. The classification accuracy of "Ref 3" requires far fewer computation resources than the "Ref 1" that mainly contains 3×5 kernels in its second convolutional layer, although both networks slightly outperform the benchmark architecture.

Fig 3.9 illustrates three optimised network architectures using Fashion-MNIST dataset. As is shown in Fig 3.9, the kernels in both convolutional layers of optimised "Ref 1" mainly involves 3×3 kernels, and sightly increases the classification accuracy over the benchmark architecture. The kernel distributions of "Ref 2" and "Ref 3" on Fashion-MNIST are quite similar to the "Ref 2" and "Ref 3" on MNIST, where the first convolutional layer mostly features 5×5 kernels and uses smaller kernels.

Fig 3.10 demonstrates how the convolutional layers of the optimised architecture are built by the proposed method using CIFAR-10 dataset. It can be seen from Fig 3.10, how the optimised architecture with the highest classification accuracy, "Ref 1", is mainly involves 3×3 kernels in both first and second convolutional layers. The "Ref 1" architecture is quite similar with most state-of-the-art CNN design methodology, such as VGG [5], which only contains 3×3 kernels in its convolutional layers. The kernels in the first convolutional layer of "Ref 2" are quite similar to the kernels in the first convolutional layer of "Ref 1", where both of them have 26 3×3 kernels with several other shapes of kernels. The second convolutional layer of "Ref 2" mainly contains 1×1 kernels, which saves around 1.5X computational resources and only has 1% classification accuracy decrease compared with "Ref 1". The third reference point, "Ref 3", has a classification accuracy of about 1.5% lower than the benchmark

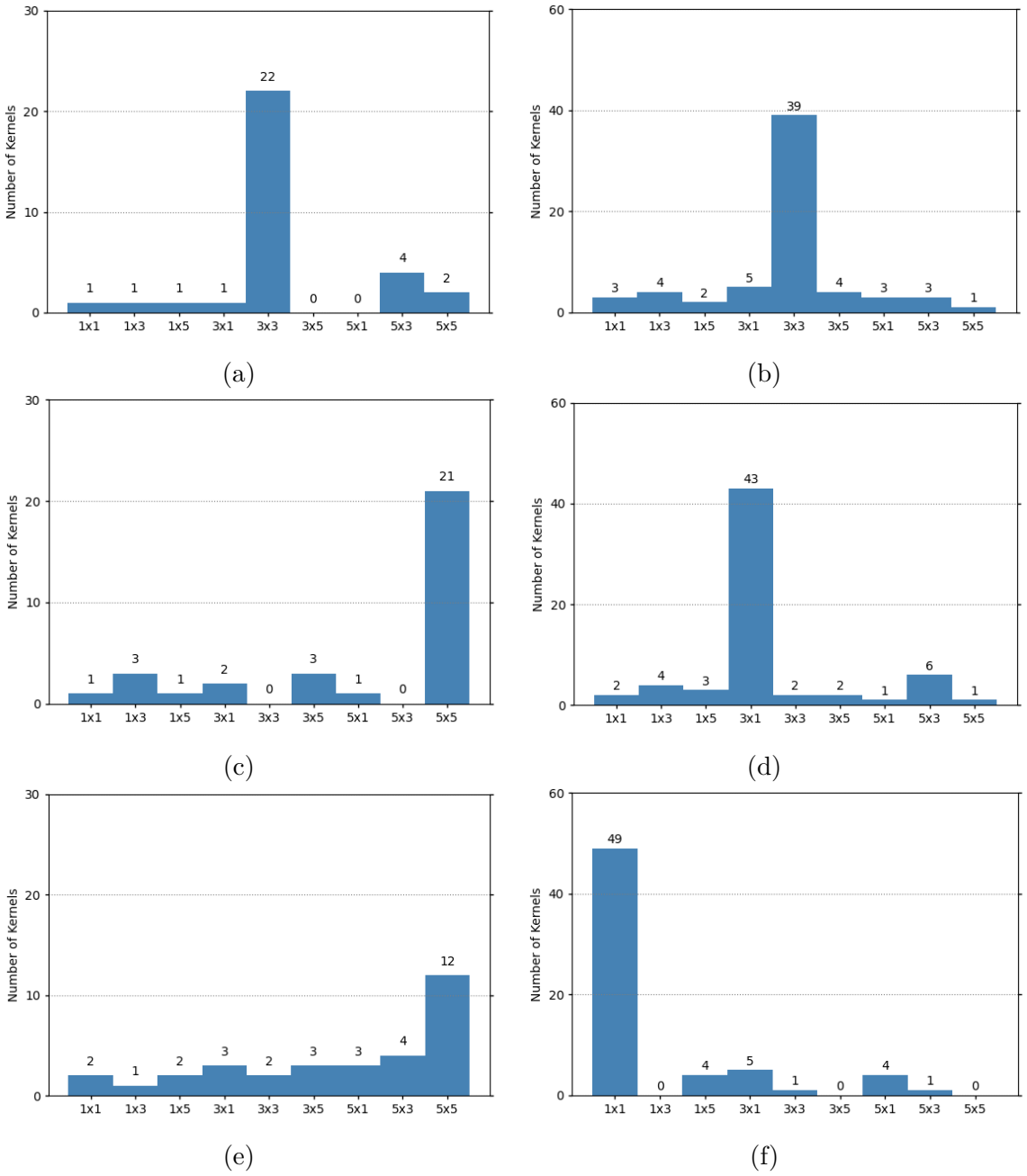


Figure 3.9 (a) Kernel distribution of the first convolutional layer of Ref 1 on Fashion-MNIST dataset. (b) Kernel dist. of the second conv. layer of Ref 1 on Fashion-MNIST dataset. (c) Kernel dist. of the first conv. layer of Ref 2 on Fashion-MNIST dataset. (d) Kernel dist. of the second conv. layer of Ref 2 on Fashion-MNIST dataset. (e) Kernel dist. of the first conv. layer of Ref 3 on Fashion-MNIST dataset. (f) Kernel dist. of the second conv. layer of Ref 3 on Fashion-MNIST dataset.

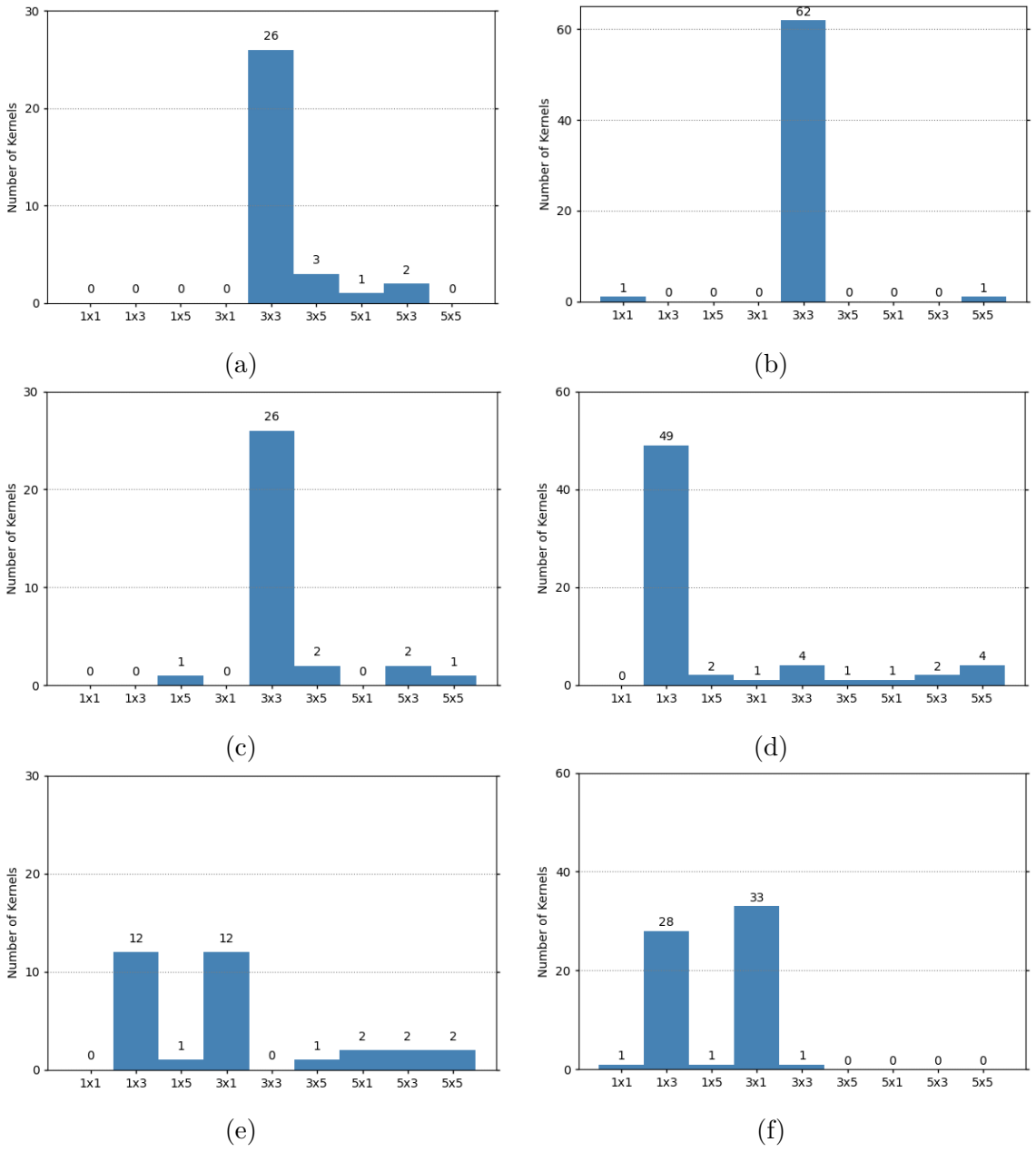


Figure 3.10 (a) Kernel distribution of the first convolutional layer of Ref 1 on CIFAR-10 dataset. (b) Kernel dist. of the second conv. layer of Ref 1 on CIFAR-10 dataset. (c) Kernel dist. of the first conv. layer of Ref 2 on CIFAR-10 dataset. (d) Kernel dist. of the second conv. layer of Ref 2 on CIFAR-10 dataset. (e) Kernel dist. of the first conv. layer of Ref 3 on CIFAR-10 dataset. (f) Kernel dist. of the second conv. layer of Ref 3 on CIFAR-10 dataset.

but consumes around 7X computational resources less than the benchmark. From the optimisation results, "Ref 3" mainly features 3×3 kernels in both convolutional layers.

3.5 Summary

In summary, this chapter 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 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) classification accuracy.

The proposed method has been tested on MNIST, Fashion-MNIST and CIFAR-10 datasets. As can be seen from the results, the proposed method shows large improvements on computational resource consumption, some optimised results with increases in classification accuracy, compared with conventional design of convolutional layers. Comparing reference points from the optimisation results on each dataset, the first convolutional layers usually involves conventional square kernels, but in the second convolutional layers, different shapes of unconventional kernels have been placed.

Chapter 4

Evolutionary Optimisation of Convolutional Layer Design

4.1 Overview

According to the results from previous investigation in Chapter 3, replacing the square convolution kernels with combinations of unconventional kernels found by MOEA can significantly reduce the number of multiplications required in convolutional layers while processing CNNs. In this chapter, a further optimisation of reducing the number of multiplications in convolutional layers is considered.

State-of-the-art CNN designs contain multiple convolutional layers in order to extract multiple features from input images. For instance, VGG-11 [5] has eight convolutional layers. The first convolutional layer of VGG-11 involves 64 convolution kernels, the second convolutional layer of VGG-11 has 128 convolution kernels, the third and fourth convolutional layer contain 256 convolution kernels each. Then, all subsequent of convolutional layers of VGG-11 involve 512 convolution kernels. The trend of increasing numbers of kernels in convolutional layers of the ResNet architecture [30] is similar to the VGG architecture, where the number of kernels in convolutional layers of ResNet increases from 64 to 512. As of yet, there is not a clear methodology that describes how many convolution kernels should be used to design convolutional layers most efficiently. To address this problem, in this chapter a multi-objective evolutionary algorithm-based method is proposed to optimise pre-designed CNN architecture. The optimisation method is proposed to use MOEA to reduce the number of kernels in convolutional layers for a specific CNN architecture. As the total number of kernels in convolutional layers decreases, the computational resources that are required for processing the CNN is reduced. At the same time, the previous chapter has verified that using a combination of unconventional kernels can significantly reduce the number of multiplications in convolutional layers, the proposed method aims to combine both reduction of size and number of kernels in convolutional layers.

In order to compare how the reduction in number of kernels in convolutional layers affects network classification accuracy, the proposed method is tested on the same

datasets and CNN architectures as the experiments in Chapter 3, that is MNIST, Fashion-MNIST and CIFAR-10 dataset with LeNet. Then, the proposed method has been tested on deeper CNN architectures, including five-layers networks, six-layers and VGG11 on CIFAR-10 dataset, to show the scalability of proposed method for optimising large-scale CNN architectures.

This chapter is organised as follows: Section 4.2 gives the details that applying MOEA to optimise and explore the design space of CNN architecture and explains its features. Section 4.3 demonstrates test experiments of proposed method with different CNN architectures on various datasets and provides a proof-of-concept of the performance of the proposed method. As well as comparing the solutions which are founded by the proposed method with other peer competitors. Finally, Section 4.4 concludes the chapter.

4.2 Methodology

In this section, an improved multi-objective optimisation method is proposed with the aim to reduce the number of computation operations further and ensure good classification accuracy of low-cost CNN models. In Chapter 3, the experimental results demonstrate that using different shapes and sizes of convolution kernels can significantly reduce the computational resource consumption and preserve, or improve, the classification accuracy for processing CNN applications. This section gives the details of the proposed method that to further reduce the computational resource consumption.

4.2.1 Multi-objective Evolutionary Algorithm

As the computational resource analysis described in Chapter 3, the number of multiplication in convolutional layers of CNNs is proportional to the size and number of convolution kernels, as shown in (3.2). Therefore, in order to reduce the computational resource consumption in processing CNNs, both size and number of convolution kernels are now minimised. In this case, NSGA-II [18] has been adopted as the optimisation tool. Specifically, the fast non-dominated sorting approach and diversity preservation strategy of NSGA-II ensure that the optimisation can converge to a uniform spread of Pareto-optimal results which can achieve a trade-off solution between multiple objectives. The Pareto-optimal solution allows the CNN architecture to achieve trade-off solutions between number of multiplications required in convolutional layers, total number of convolution kernels, and the classification accuracy on the task dataset.

4.2.2 Convolutional Layer Optimisation

The convolutional layer evolutionary optimisation framework is illustrated in Fig 4.1. The optimisation framework aims to minimise the number of kernels in convolutional layers while simultaneously applying unconventional kernels introduced in Chapter 3.

The proposed multi-objective optimisation flow involves the following components:

- **Parametric convolution kernels:** The pre-designed CNN involves different numbers and sizes of kernels in each convolutional layer. The MOEA representation encodes the size of kernels into a set of genes. An integer representation is used to represent the size of kernels, for instance, 1 represents a kernel size of 1×1, 2 represents the kernel size of 1×3, 3 represents the kernel size of 1×5, 4 represents the kernel size of 3×1 and so on, as the set of unconventional kernels shows in Fig 3.2a. In this case, the number '0' represents 'removing kernel'. After

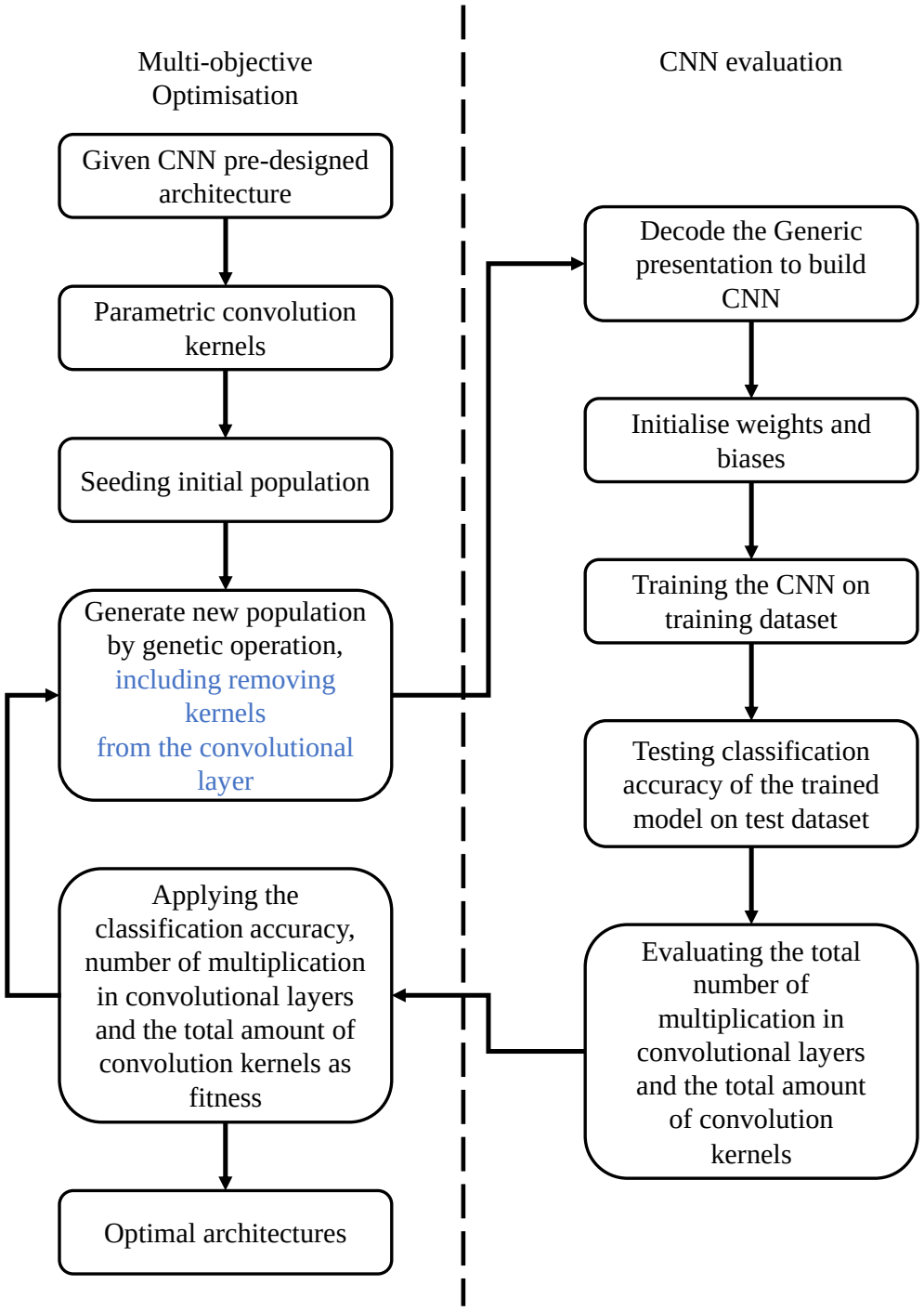


Figure 4.1 Convolutional layer optimisation flow. On the left hand side are the evolutionary algorithm steps. On the right hand side is the CNN evaluation process.

parameterising convolution kernels into genes, the total number of genes in each individual will be the sum of kernels in each convolutional layer.

- **Seeding the initial population:** In this work, the initial populations are seeded by manually setting several sets of unconventional kernels, randomly generating combinations of unconventional kernels, and setting a pre-designed architecture. In particular, the first individual represents the original pre-designed network, such LeNet [27] and VGG [5]. From the previous experimental results in Chapter 3, the architectures found by MOEA contain large number of a single type of convolution kernels in each layer, combining with small number of other type of convolution kernels. Therefore, from the second individual to 10th individual, they are set to single type of unconventional kernels from 1×1 to 5×5, respectively. Finally, the rest of individuals are seeded by random combination of unconventional kernels.

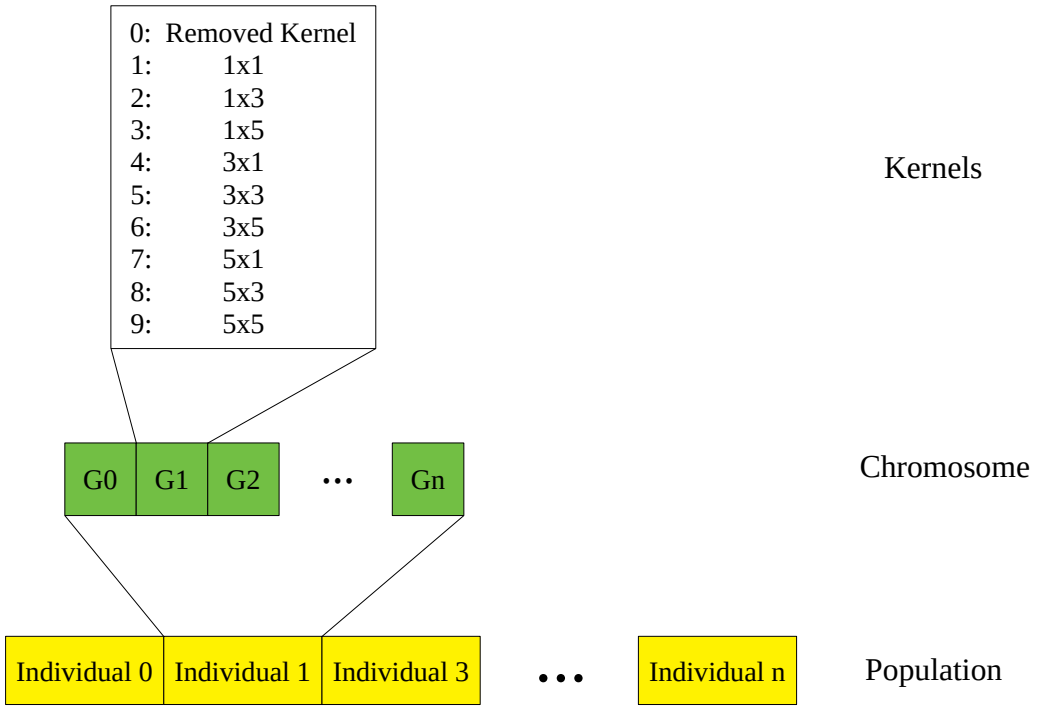


Figure 4.2 An example that describes the population and chromosome for the proposed method.

- **Genetic Operator:** In the proposed method, only mutation operation is used as genetic operator. The mutation operation modifies the size and shape of convolution kernels based on a given mutation probability. The mutation probability is defined by $\mathbf{p}$, with $\mathbf{p} \in [0, 1]$. The mutation operation generates a new network architecture that is ready for training and evaluation. In this case, the mutation process is presented in Algorithm 3.

Algorithm 3 Mutation operation

Procedure: Mutation (P, ρ). ▷ Population (P) has N individuals and M genes in each individual. The mutation probability is ρ.

```
Offspring population (O) ← P
for i ← 1 to N do
    for j ← 1 to N do
        if random(0, 1) < ρ then
            O_{i,j} ← randomInt(0, 9)
        end if
    end for
end for
```

- **Fitness evaluation:** The fitness evaluation calculates the fitness scores of each individual. In the proposed method, there are three objectives that need to be evaluated in the MOEA. The first objective is the network classification error. The classification error is defined by (4.2.2). The second objective is the total number of multiplications in convolutional layers which refers to overall computational cost for processing the CNN. The number of multiplications in convolutional layers, $Operation_{total}$, is calculated by the sum of number of multiplications in each convolutional layer, as described :

$$Operation_{total} = \sum_{l=1}^{L}(\sum_{c=1}^{C}(\sum_{n=1}^{N}(F_h \times F_w \times K_{hn} \times K_{wn}))) \tag{4.1}$$

where L describes the number of convolutional layers in the target CNN. C is the number of input feature maps. N is the number of convolution kernels. F_h and F_w indicate the height and width of the input feature maps respectively. K_{hn} and

K_{wn} indicate the height and width of convolution kernels. The third objective is the total number of convolutional kernels in the target CNN. Therefore, the fitness function for minimising three objectives is described in (4.3).

$$Classification\ Error = 1 - Classification\ Accuracy \tag{4.2}$$

$$f(x) = \min\ [classification\ error,\ No.\ of\ multiplications,\ No.\ of\ kernels] \tag{4.3}$$

The adapted algorithm for optimising the convolutional layers, in this case, is demonstrated in Algorithm 4.

Algorithm 4 NSGA-II for optimising convolutional layer

Procedure: NSGA-II (M, N, $f(x)$). ▷ evolving N individuals for M generation with fitness function of $f(x)$, $\forall x \in X$, where X is the set of convolution kernels.

1: Initialise parent population $P_1 = [x_1,\ x_2,\ ...,\ x_N]$
2: Offspring population $(O_1)\ \leftarrow$ Mutation(P_1)
3: **for** $i \leftarrow 1$ *to* M **do**
4: $C_i\ \leftarrow\ O_i \cup P_i$ in size $2N$
5: $R_i\ \leftarrow\ f(C_i)$ ▷ evaluating each individual in C_i with fitness function $f(x)$
6: $F\ \leftarrow$ Non-Dominated-Sorting(R_i)
7: $P_{i+1}\ \leftarrow \emptyset$
8: $j \leftarrow 1$
9: **while** $|P_{i+1}|\ +\ |F_j|\ \leq\ N$ **do**
10: Crowding-Distance-Calculation(F_j)
11: $P_{i+1}\ \leftarrow\ P_{i+1} \cup F_j$
12: $j\ \leftarrow\ j + 1$
13: **end while**
14: $F_j\ \leftarrow$ Descend-Sort(F_j)
15: $P_{i+1}\ \leftarrow\ P_{i+1} \cup F_j[1 : (N - |P_{i+1}|)]$ ▷ individual with less crowding distance from the first to the $(N - |P_{i+1}|)th$ of F_j to fill P_{i+1}.
16: $O_{i+1}\ \leftarrow$ Mutation(P_{i+1})
17: **end for**

4.3 Experimental Setup and Results

In this section, the proposed method is tested and evaluated on various CNN architectures with multiple datasets. The aim in this section is to show the performance gain of the proposed compared with conventional CNN design in terms of trade-off between network's classification accuracy, computational costs and number of convolution kernels.

4.3.1 Experiment Setup

The experiments are set out in two parts. The first part is testing the proposed method with LeNet, the same as the benchmark architecture in Chapter 3, on MNIST, Fashion-MNIST and CIFAR-10 datasets. In this part, the set of experiments is aimed to find out how the reduction in number of convolution kernels can further improve the network performance, compared with only using unconventional shapes and sizes of kernels as proposed in Chapter 3. The benchmark CNN is trained on training datasets, which contain 60,000 28×28 grey-scale images on MNIST and Fashion-MNIST, and 50,000 32×32 RGB images on CIFAR-10 associated with a label from 10 different categories. All of the benchmarks are trained by Stochastic Gradient Decent (SGD) method using Adam optimiser [90] with an initial learning rate of 1e-3 for 100 epochs. The learning rate is reduced by factor of 10 at 30th epoch. The softmax cross-entropy loss is used as the loss function. In order to improve the classification accuracy by reducing network overfitting, L_2 regularisation [92] with a weight decay of 0.0001 is applied to the training process. A dropout rate of 0.5 in fully-connected layer is also applied to the training process. Then, the benchmark accuracy for each dataset is reported by evaluating trained models on test datasets, that involve 10,000 28×28 grey-scale images on MNIST and Fashion-MNIST, and 10,000 32×32 RGB images on CIFAR-10.

The second part of the experiments is testing the proposed method with three deep CNNs, including five-layer, six-layer CNNs and VGG-11 [5] on CIFAR-10 dataset. In the experiments, two CNNs are constructed by ourselves, i.e. five-layer CNN and six-layer CNN. These two CNNs are designed based on the LeNet architecture but deeper than the original LeNet. The five-layer CNN adds an extra convolutional layer after the second convolutional layer of LeNet with 64 3×3. The six-layer CNN adds two convolutional layers after the second convolutional layer of LeNet, both two extra layers have 64 3×3 convolutional kernel in each of them. The reason for constructing the five-layer CNN and six-layer CNN in this way is because that there is a big gap of depth between LeNet and VGG-11. Therefore, we construct the five-layer CNN and six-layer CNN to evaluate the proposed method in multiple CNN architectures.The second part of experiments aims to demonstrate that the proposed method is capable of optimising various deeper CNN architectures. The three CNNs are:

- **Five-layer CNN:** The five-layer benchmark architecture consists of three convolutional layers, each layer includes 64 3×3 convolution kernels. Each convolutional layer is followed by a 2×2 max pooling operation. A fully-connected layer is connected to the last max-pooling layer with 512 neurons. The classification layer is connected to the end.

- **Six-layer CNN:** The six-layer benchmark architecture has four convolutional layers, each layer also includes 64 3×3 convolution kernels. The max pooling operations are applied at the end of the first, second and fourth convolutional layers. After the last max-pooling operation, there is a fully-connected layer with a total of 512 neurons and a sofmax operation is used to predict the classification at the classification layer.

- **VGG-11:** The VGG-11 was designed by K. Simonyan and A. Zisserman [5]. This model totally contains eight convolutional layers, two fully-connected layers and one classification layer. All of kernels in each convolutional layer have dimension

of 3×3. The first convolutional layer of VGG-11 has 64 kernel and the second convolutional layer has 128 kernels. The third and fourth convolutional layer involve 256 kernel respectively. Then, there are 512 kernels designed for 5th, 6th, 7th and 8th convolutional layer respectively. The max pooling operation is activated at the end of 1st, 2nd, 4th, 6th and 8th convolutional layers. Then, both 1st and 2nd fully-connected layer have 4096 neurons in each one. The original VGG-11 is designed for using in ImageNet [10] dataset, which has 1000 labels. In the experiment, the VGG-11 is targeted to handle CIFAR-10 dataset, that is 10 labels in total. Therefore, in this case, the classification layer of VGG-11 is modified to contain 10 neurons.

For these deeper CNNs, experiments are implemented on CIFAR-10 dataset. The benchmarks have been trained by SGD method with Adam optimiser [90] for an initial learning rate of 1e-3 for 100 epochs. The learning rate is reduced by factor of 10 at 30th epoch. Softmax cross-entropy loss has been used as the loss function. To prevent overfitting, L_2 regularisation [92] with a weight decay of 0.0001, dropout and data augmentation methods are applied to the training process. The data augmentation used is based on [91], that is padding 4 pixels on each side and randomly crop a patch from the padded image or its horizontally flipped version. A dropout rate of 0.5 is applied to the training process.

In this section, all of the optimisation loops have been set to run for 100 generations for a population size of 25 individuals and only mutation operation has been used as genetic operator. The mutation rate of all experiments is set to 0.1. All experiments are implemented on Intel Xeon 6138 20-core 2.0 GHz CPU with single NVIDIA Tesla V100 32GB SXM2 GPU. The approximate runtime of each experiment is reported on Table 4.1.

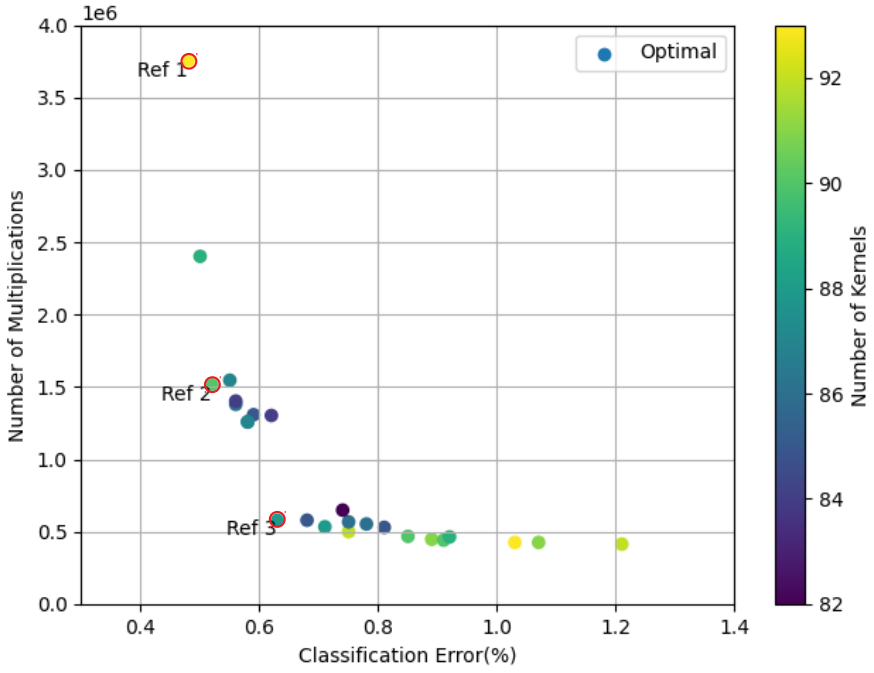
Table 4.1 Approximate runtime of the optimisation loop on each dataset.

Model	Dataset	Runtime
LeNet-5	MNIST	$\approx$ 5 days
	Fashion-MNIST	$\approx$ 5 days
	CIFAR-10	$\approx$ 9 days
Five-layer CNN	CIFAR-10	$\approx$ 12.5 days
Six-layer CNN	CIFAR-10	$\approx$ 16.6 days
VGG-11	CIFAR-10	$\approx$ 33.5 days

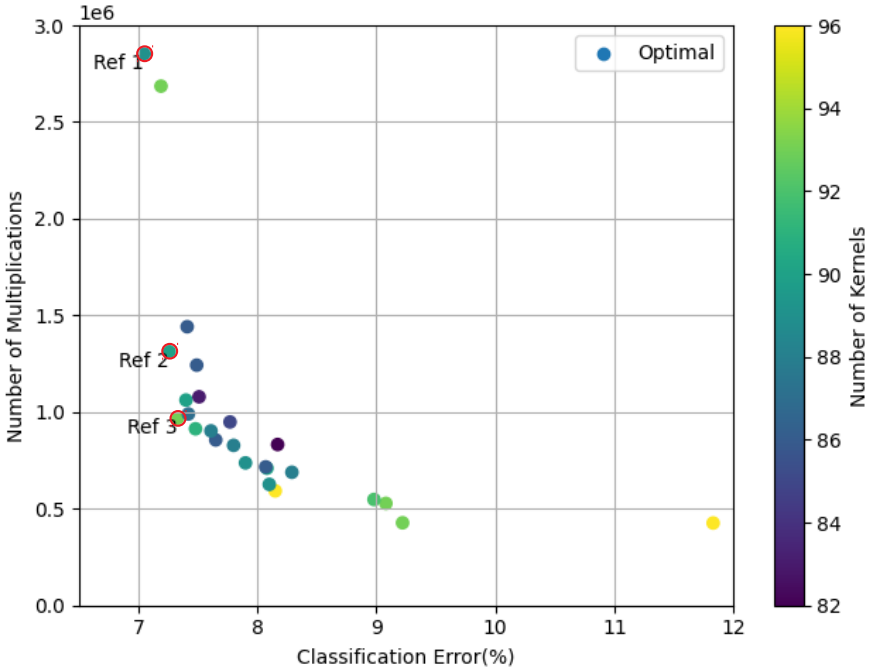
4.3.2 Experimental Results on LeNet

For evaluating the proposed method on MNIST and Fashion-MNIST, the LeNet is configured with 28×28 neurons in input layer. The initial population are seeded with 25 individuals. The first individual is the original test architecture, that contains 32 5×5 kernels in first convolutional layer and 64 5×5 kernels in second convolutional layer. From the second to the 9th individuals, they are seeded with single type of kernels from 1×1 to 5×3, as described in Section 4.2. Rest of individuals are initialised with random kernels. Each individual are then trained by SGD method for 100 epochs with learning rate of 1e-3. The proposed method is set to run for 100 generations and optimisation results are reported by the last generation.

Fig. 4.3 illustrates the optimisation results for LeNet on MNIST and Fashion-MNIST after running 100 generations following the same methodology from Chapter 3. As shown in Fig 4.3, three reference points from the optimisation results have been selected for comparing the network performance with original architecture for each dataset. The first reference point is the network architecture with the highest classification accuracy from the set of optimal results. The second reference point uses less resource than first reference architecture while keeping high classification accuracy. The third reference point is the trade-off solution that closing the the origin between the network classification accuracy, the number of multiplications and the number of kernels.



(a)



(b)

Figure 4.3 (a) Optimisation results for proposed method after running 100 generations on MNIST. (b) Optimisation results for proposed method after running 100 generations on Fashion-MNIST.

The benchmark CNN achieves a classification accuracy of 98.92% on MNIST dataset and 92.12% on Fashion-MNIST. Both benchmark networks require a total of 10,662,400 multiplications in their convolutional layers. The comparison between selected reference architectures and benchmarks on each dataset is shown in Table 4.2.

Table 4.2 Comparison between the benchmark network and solutions found by proposed method on three datasets. Three reference points are selected from optimised results for each dataset.

Dataset	Model	Top-1 Acc.	Acc. improve	Reduction in Mults.
MNIST	Benchmark	98.92%	-	-
	Ref 1	99.52%	0.60%	2.84x
	Ref 2	99.48%	0.56%	7.04x
	Ref 3	99.38%	0.46%	18.38x
Fashion-MNIST	Benchmark	92.12%	-	-
	Ref 1	92.96%	0.84%	3.74x
	Ref 2	92.75%	0.63%	8.12x
	Ref 3	92.67%	0.55%	11.03x

As shown in Table 4.2, all reference points from the optimal solutions that are found by the proposed method demonstrate significant reduction in number of multiplications while the network classification accuracy is maintained, or outperforms the benchmark architecture on both MNIST and Fashion-MNIST datasets. The reference points selected here are following the same selection methodology as in Chapter 3. The results for all of the reference point solutions in this experiment feature lower computational resource costs than the reference points that are obtained by two-objective optimisation method in Chapter 3.

Fig 4.4 and Fig 4.5 illustrate the kernel distributions of the three reference architectures that are found by the proposed method on MNIST and Fashion-MNIST dataset, respectively. Compared with the results in Chapter 3, the three objective optimisation method reduces the number of convolution kernels in all case.

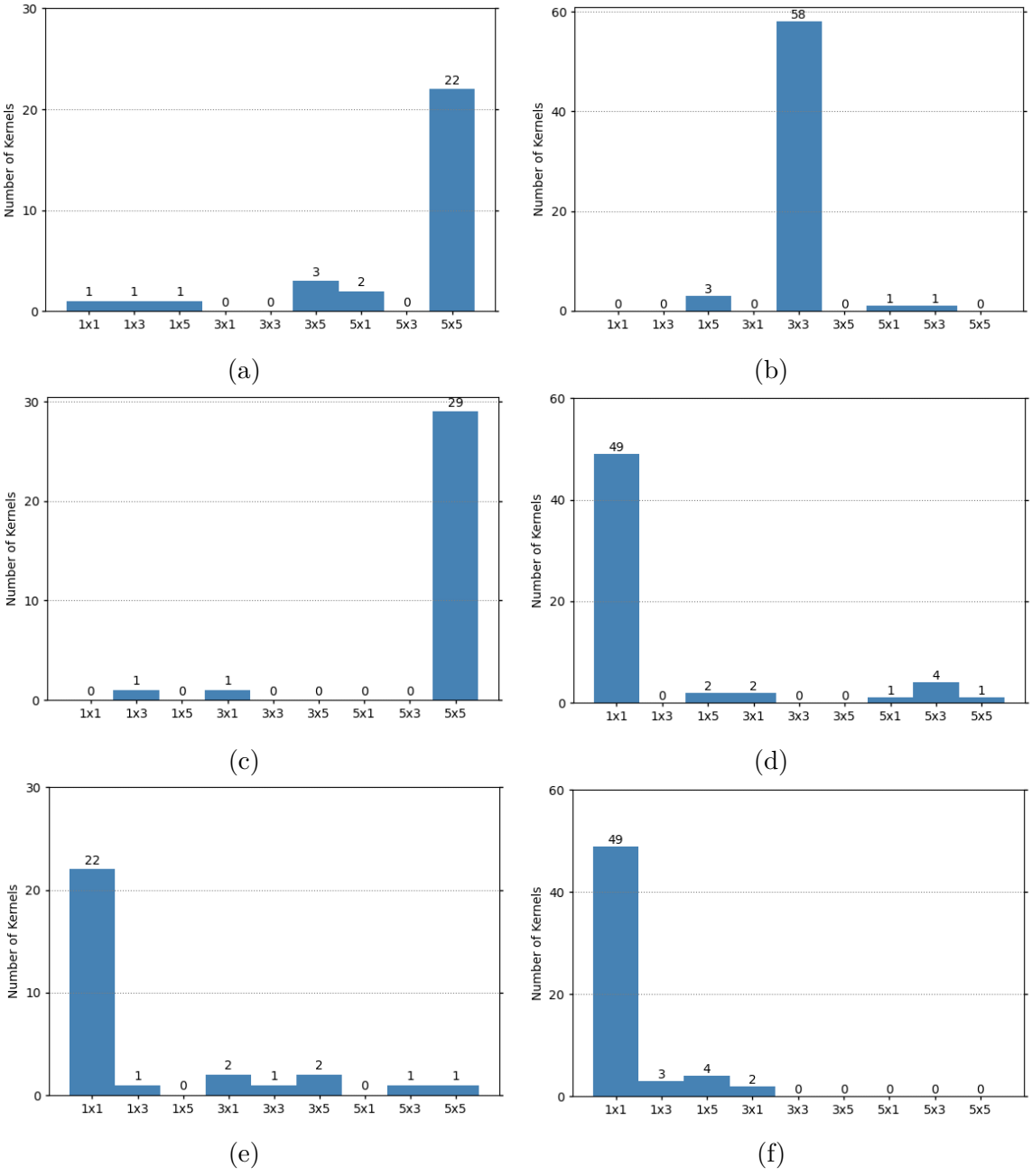


Figure 4.4 The figure demonstrates the kernels that are built up the CNNs by the proposed method on MNIST dataset. (a) Kernel distribution of the first convolutional layer of Ref 1 on MNIST dataset. (b) Kernel dist. of the second convolutional layer of Ref 1. (c) Kernel dist. of the first convolutional layer of Ref 2. (d) Kernel dist. of the second convolutional layer of Ref 2. (e) Kernel dist. of the first convolutional layer of Ref 3. (f) Kernel dist. of the second convolutional layer of Ref 3.

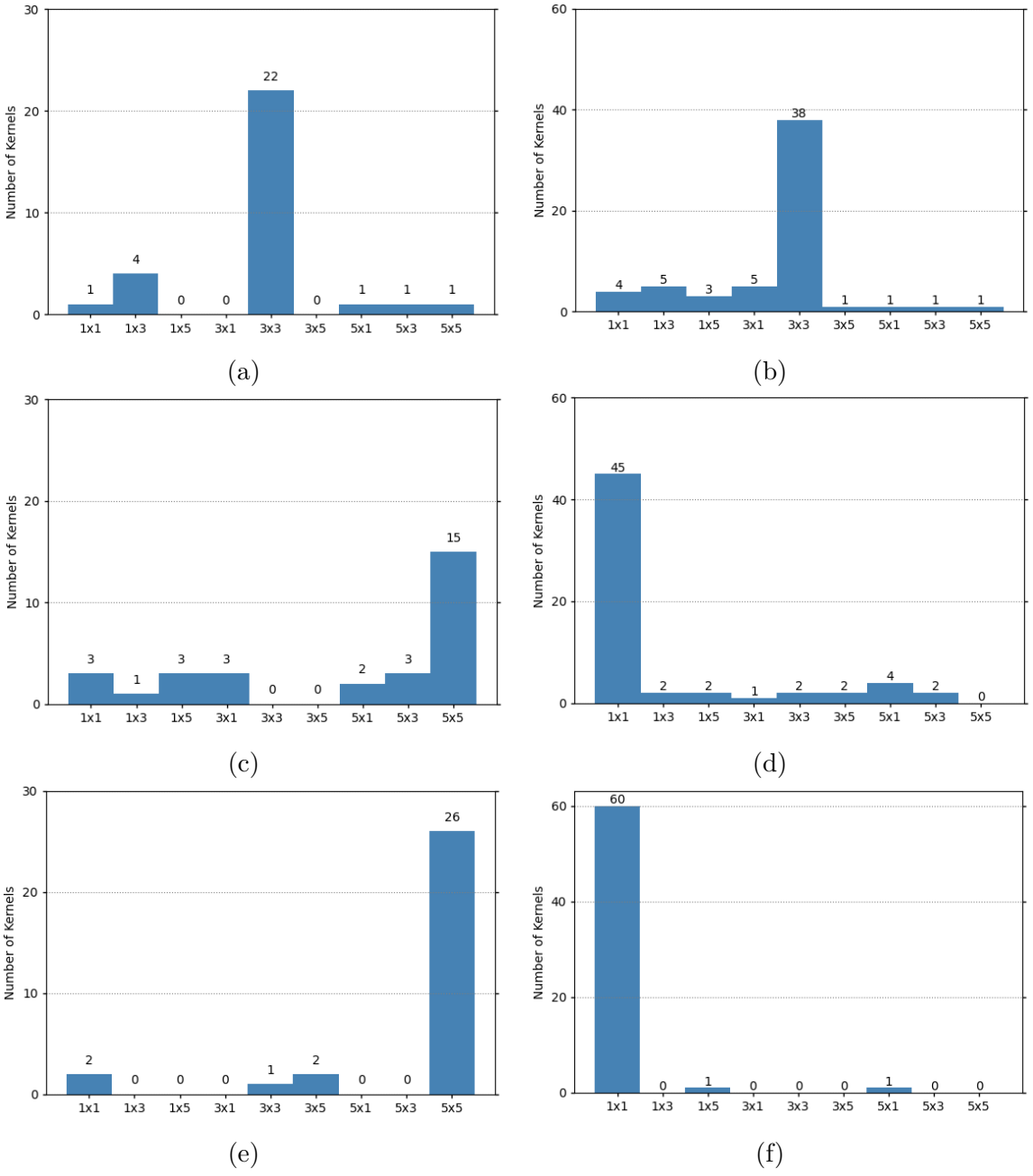


Figure 4.5 The figure demonstrates the kernels that are built up the CNNs by the proposed method on Fashion-MNIST dataset. (a) Kernel distribution of the first convolutional layer of Ref 1 on Fashion-MNIST dataset. (b) Kernel dist. of the second convolutional layer of Ref 1. (c) Kernel dist. of the first convolutional layer of Ref 2. (d) Kernel dist. of the second convolutional layer of Ref 2. (e) Kernel dist. of the first convolutional layer of Ref 3. (f) Kernel dist. of the second convolutional layer of Ref 3.

Then, the proposed method has been tested on LeNet with CIFAR-10 dataset. The CIFAR-10 dataset contains totally 50,000 training images and 10,000 test images. In this experiment, only the training images from CIFAR-10 dataset are used for the optimisation loop. The 50,000 images are split into two parts, sub-training set and sub-validation set. The sub-training set contains 40,000 and the sub-validation set has 10,000 images. The sub-training set is used for training CNNs which are generated by the MOEA and the sub-validation set is used for testing the classification for these CNNs. After the optimisation process, optimal results are then re-trained and tested on full CIFAR-10 dataset.

In order to handle RGB images, the LeNet has been modified with 3-channel inputs. In this case, the $L2$ regularisation of 0.0001 and data augmentation methods [91] have been applied for training the CNNs. For the data augmentation, input images are randomly padded 4 pixels on each side and randomly cropped a patch from the padded images or its horizontally filliped version. After training the network for 100 epochs, the benchmark requires 15,564,800 multiplication operations with testing classification error of 17.63% on CIFAR-10.

For optimising the LeNet on CIFAR-10 dataset, the population has been set to 25 individuals with mutation rate of 0.1. After running for 100 generations, the result has been shown in Fig 4.6. There are three reference points from the last generation been selected as optimal architectures. The first reference point is the network architecture which features the highest classification accuracy achieved by the proposed method. The second reference point is the one that still features good classification performance while uses less computational resources than the first reference point. Notably, it also outperforms the benchmark network architecture in this case. The third reference point is the best trade-off (closest to origin) between number of multiplications, number of kernels and classification accuracy after optimisation.

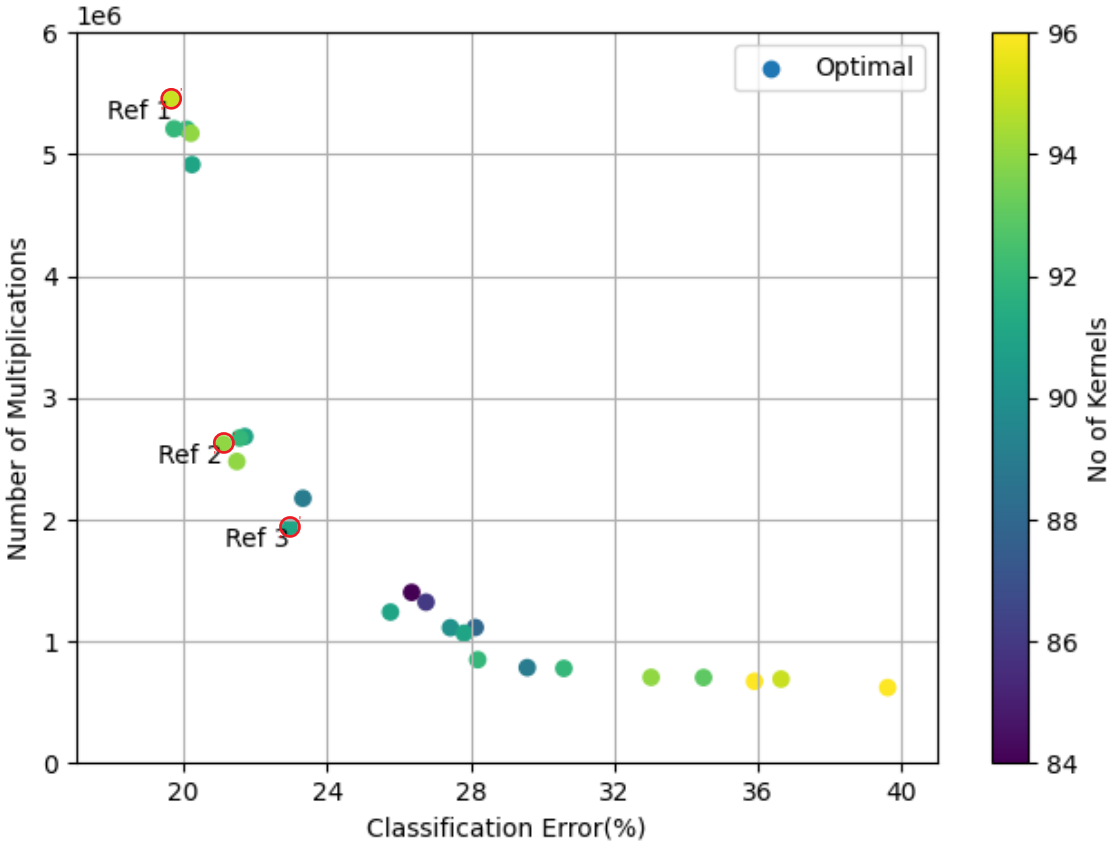


Figure 4.6 Optimised architectures after 100 generations on CIFAR-10 by the proposed method.

Then, these three reference architectures have been re-trained on the full CIFAR-10 dataset, i.e. 50000 training images, by the same methodologies as the benchmark. After training for 100 epochs, three reference architectures have been tested on the testing set, i.e. 10000 testing images. The comparison of networks performance between the benchmark and reference points after re-training is shown in Table 4.3.

Table 4.3 Comparison between the benchmark network and solutions found by proposed method on CIFAR-10 dataset with three-objective optimisation.

Dataset	Model	Top-1 Acc.	Acc. improve	Reduction in Mults.
CIFAR-10	Benchmark	82.37%	-	-
	Ref 1	83.47%	1.10%	2.85x
	Ref 2	82.46%	0.09%	5.93x
	Ref 3	80.73%	-1.64%	8.03x

Fig. 4.6 illustrates optimal solutions for the benchmark network on the CIFAR-10 dataset. Compared with the two-objective optimisation in the Chapter 3, a number of kernels in both convolutional layers have been removed by the three-objective optimisation method. By removing some of the kernels, optimising for the three objectives can further reduce the number of multiplications that are required to process the CNN without compromising significantly on classification accuracy. In addition, Fig. 4.7 shows the kernel distributions of three reference points for both convolutional layers after applying the proposed method. It can be seen from Fig. 4.7, Ref 1 contains a majority of 3×3 kernels, and combined with a small number of unconventional kernels in both convolution layers. For Ref 2 and Ref 3, these networks mainly involve 3×3 kernels in the first convolutional layer and also make use of kernels with different sizes. In the second convolution layer of Ref 2, most of 3×3 kernels are replaced by 1×3 kernels and 3×1 kernels. This replacement is the main reason for Ref 2's significant reduction in computational resource compared Ref 1 and only has a small impact on its classification accuracy. Both the Ref 1 and Ref 2 outperform the benchmark CNN. Ref 3 provide the best trade-off between the number of multiplications, number of kernels

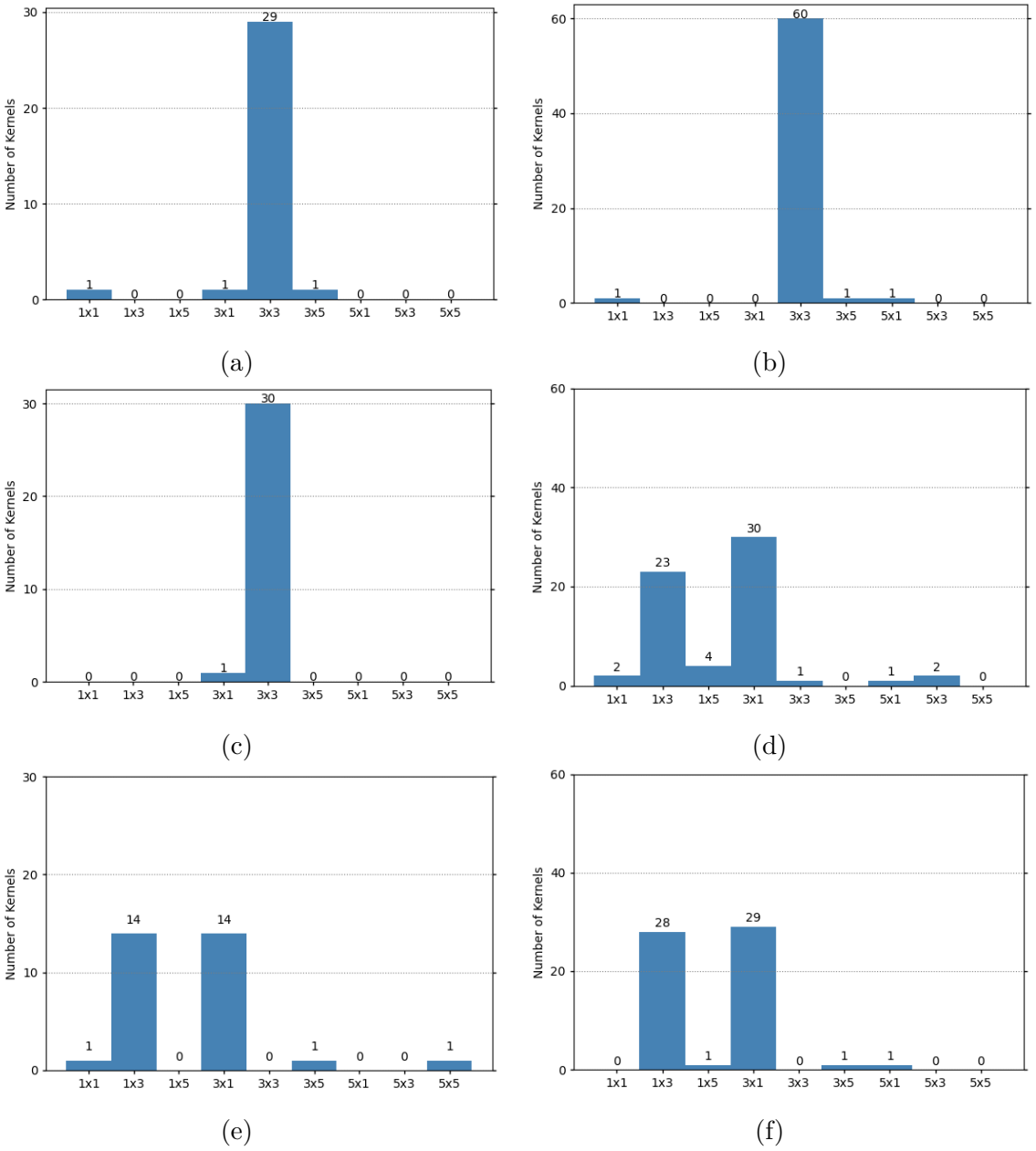


Figure 4.7 The figure demonstrates the kernels that are built up the CNNs by the proposed method on Fashion-MNIST dataset. (a) Kernel distribution of the first convolutional layer of Ref 1 on CIFAR-10 dataset. (b) Kernel dist. of the second convolutional layer of Ref 1. (c) Kernel dist. of the first convolutional layer of Ref 2. (d) Kernel dist. of the second convolutional layer of Ref 2. (e) Kernel dist. of the first convolutional layer of Ref 3. (f) Kernel dist. of the second convolutional layer of Ref 3.

and classification accuracy. It can be seen that Ref 3 uses 8.03x fewer multiplications than the benchmark CNN with only 1.64% increases of classification error. Both convolutional layers in Ref 3 involve 1×3 kernels and 3×1 kernels at the most.

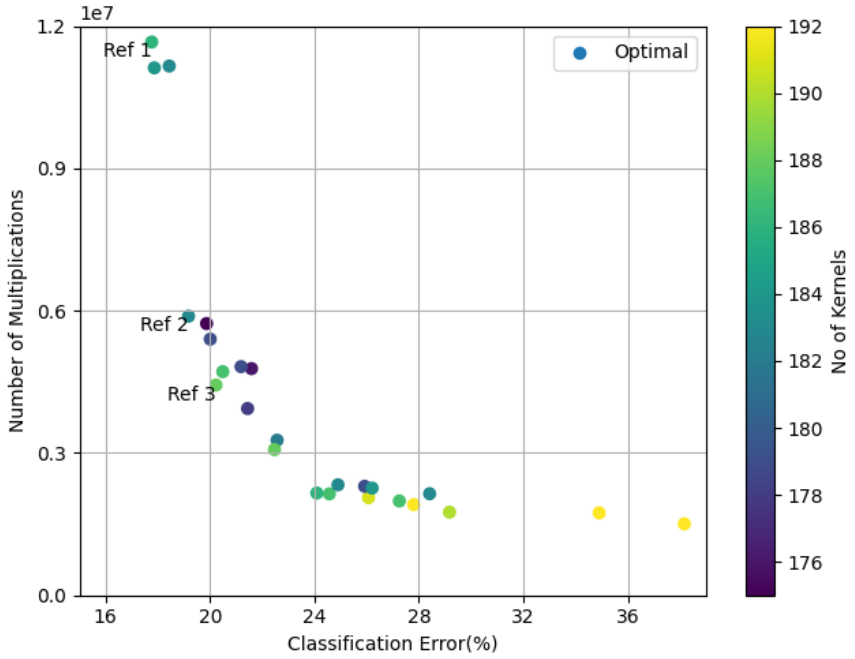
4.3.3 Experimental Results on Deeper CNNs

As shown in the results of previous experiments, the proposed method demonstrates significant improvements in reducing computational resource consumption in convolutional layers. The effectiveness of the proposed method is now evaluated on deeper CNNs to give some indication of scalability. The benchmark architecture that is used in these experiments are five-layer CNN, six-layer CNN and VGG-11 [5].

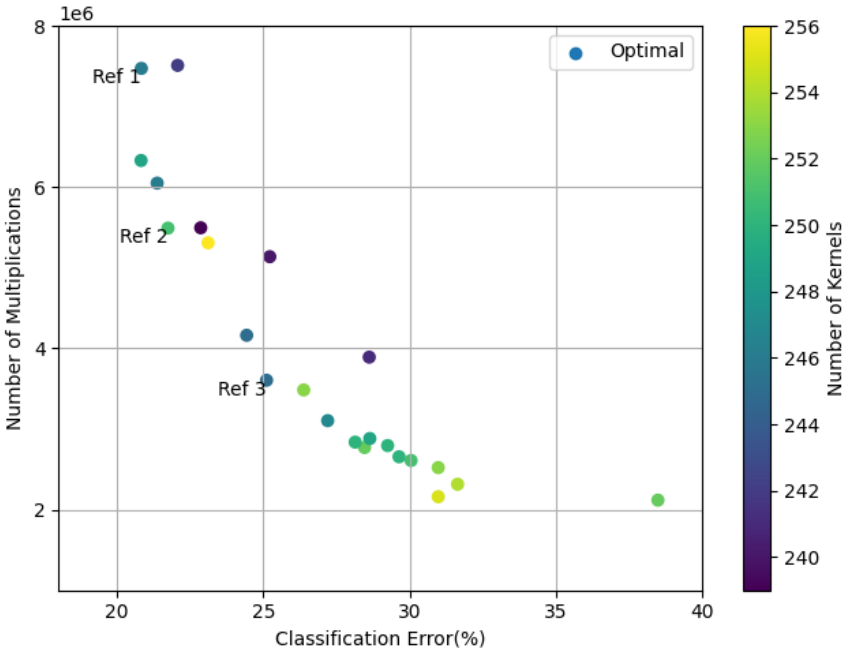
These experiments use three objectives to evaluate the fitness, which are number of multiplications, Top-1 classification error and number of convolution kernels. During optimisation process, all networks are trained on CIFAR-10 dataset for 30 epochs and the evolutionary loop is set to run 100 generations with population size of 25. There are 50,000 images used to train and evaluate each individuals. 40,000 images are randomly sampled from the training set of CIFAR-10 dataset. The rest of 10,000 from the the training set of CIFAR-10 dataset are then used for evaluation.

The $L2$ regularisation of 0.0001 and data augmentation methods [91] have been applied for training all of CNNs in these experiments. The data augmentation process includes that randomly padding 4 pixels on each side of input images and randomly cropping a patch from the padded images or its horizontally flipped version.

The Fig. 4.8 shows the optimal results that are found by the proposed method on CIFAR-10 dataset. Three reference architectures from each experiment are selected and re-trained for 100 epochs on the whole training set of CIFAR-10 and the classification accuracy is evaluated on the test set. The first reference point features the highest classification accuracy in the solution set after optimisation. The second reference



(a)



(b)

Figure 4.8 (a) Five-layer CNNs that optimised by the proposed method after 100 generations on CIFAR-10. (b) Six-layer that optimised by the proposed method after 100 generations on CIFAR-10.

point involves fewer multiplications and still features good classification accuracy, compared with benchmark network. The third reference point is the best trade-off solution (closest to origin) between number of multiplications, number of kernels, and classification accuracy. The details of reference points from each network architecture have been demonstrated in Fig 4.9 and Fig 4.10.

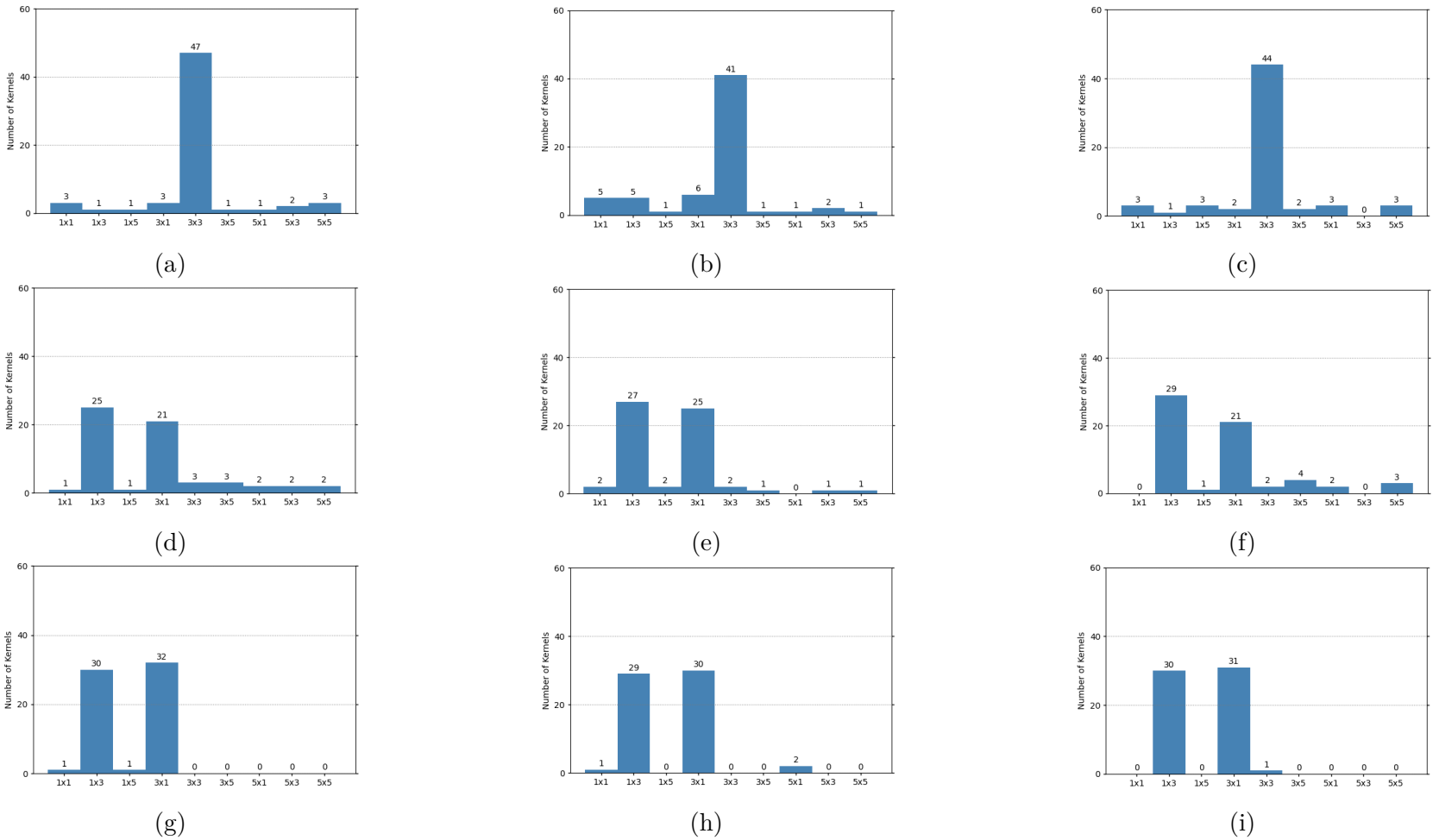


Figure 4.9 The figure demonstrates the kernels that are built up the five-layer CNNs by the proposed method on CIFAR-10 dataset. (a) to (c) shows the kernel distribution of three convolutional layers of Ref 1 from network optimisation results. (d) to (f) shows the kernel distribution of three convolutional layers of Ref 2 from network optimisation results. (g) to (i) shows the kernel distribution of three convolutional layers of Ref 3 from network optimisation results.

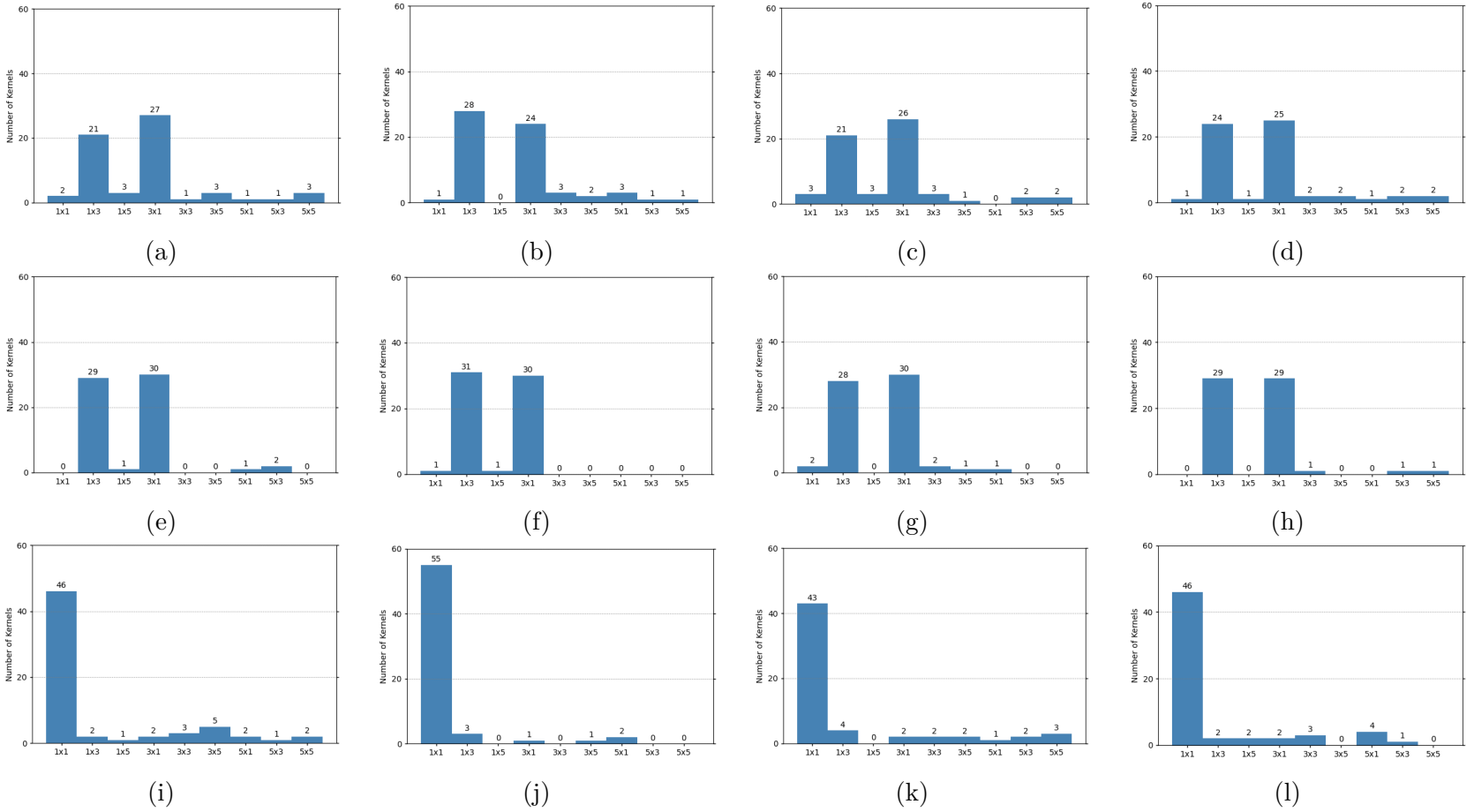


Figure 4.10 The figure demonstrates the kernels that are built up the six-layer CNNs by the proposed method on CIFAR-10 dataset. (a) to (d) shows the kernel distribution of four convolutional layers of Ref 1 from network optimisation results. (e) to (h) shows the kernel distribution of four convolutional layers of Ref 2 from network optimisation results. (i) to (l) shows the kernel distribution of four convolutional layers of Ref 3 from network optimisation results.

Table 4.4 Experimental results for five-layer CNN and six-layer CNN on CIFAR-10

Model	Referent Network	Acc. improve	Reduction in Mults.
five-layers	Ref 1	0.16%	1.17x
	Ref 2	-1.03%	2.31x
	Ref 3	-2.44%	3.06x
six-layers	Ref 1	-0.32%	2.13x
	Ref 2	-0.95%	2.90x
	Ref 3	-2.46%	4.42x

Table. 4.4 illustrates the results of three reference architectures that are found by the proposed method for each benchmark. As shown in Table. 4.4, the proposed method still can outperform the network using only conventional 3×3 convolutions, which is the standard setting in modern CNN architecture design. As the results show, the number of multiplications in convolutional layers of the three-layer CNN achieves 1.17x less than the benchmark architecture, while the Top-1 accuracy increases by 0.16%. For LeNet, the proposed method achieve 2.13x saving in number of multiplication with 0.32% decrease in Top-1 accuracy. This demonstrates the capability of the proposed method to be applied to deeper CNN architectures.

Finally, the proposed method has been applied to VGG-11 [5]. The original VGG-11 has been initially trained by SGD method with Adam optimiser [90] for an initial learning rate of 1e-3 for 100 epochs on CIFAR-10. The learning rate is reduced by factor of 10 at 30th epoch. The data augmentation methods [91], $L2$ regularisation of 0.0001 and dropout rate of 0.5 have been applied in the training process. After training process, the model achieves a classification accuracy of 86.78% on CIFAR-10 testing dataset and the number of operations in convolutional layers that required for processing CIFAR-10 images is 152,764,416.

For the optimisation loop, a total of 50,000 training images of CIFAR-10 have been split into a sub-training set of 40,000 images and a sub-testing set of 10,000 images.

Each individual has been trained on the sub-training set of 30 epochs with the same training methods as the benchmark. Then, each individual is tested on the sub-testing set. The classification accuracy on sub-testing set, number of multiplications and number of convolution kernels of each individual are reported as the fitness to the MOEA. A population size of 25 individuals with a mutation rate of 0.1 has been set to the optimisation loop. After running the proposed method for 100 generation, the optimisation results have been shown in Fig 4.11.

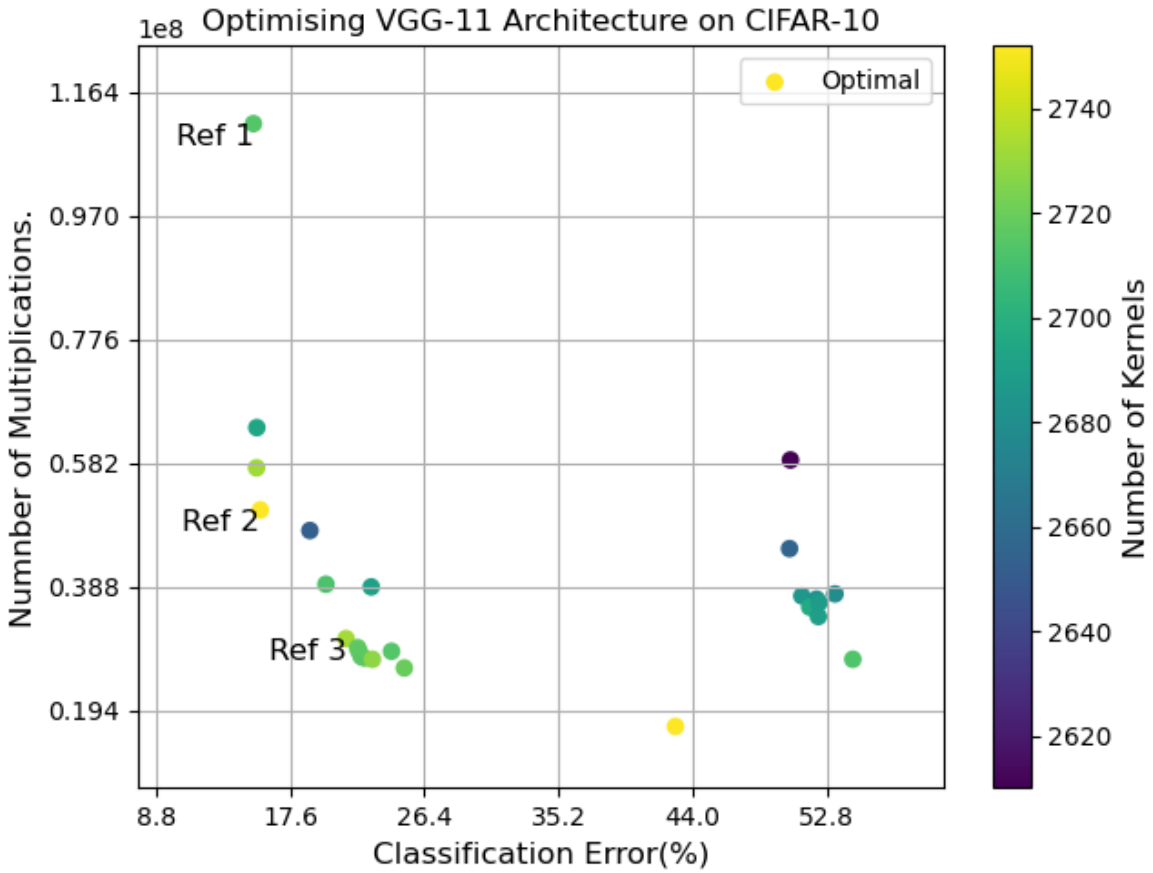


Figure 4.11 Optimisation results for proposed method after running for 100 generations on CIFAR-10.

Three reference points have been selected from the optimisation results. The first reference point, Ref 1, has the lowest classification error from the optimisation results. The classification accuracy of the second reference point, Ref 2, is close to the Ref 1, but use less computational resources. The third reference point, Ref 3, is defined as the

best trade-off between classification error, total number of multiplications and total number of convolution kernels. Then, the three reference point have been trained on the whole training dataset, i.e. 50,000 training images in total. Each reference point has been trained by SGD method for 100 epochs with Adam optimiser [90] and the initial learning is 0.001. Then, the learning rate is reduced by factor of 10 at 30th epochs. To avoid overfitting, $L2$ regularisation of 0.0001 and dropout rate of 0.5 are applied to training process. A data augmentation method [91] that padding with 4 pixels and random crop with random horizontal flip is also applied to the training process. After re-training these reference points, the comparison between the original VGG-11 and optimised solutions is shown in Table 4.5.

Table 4.5 Comparison between the original VGG-11 and solutions found by proposed method on CIFAR-10 dataset with three-objective optimisation.

Dataset	Model	Top-1 Acc.	Acc. improve	Reduction in Mults.
CIFAR-10	Benchmark	86.78%	-	-
	Ref 1	87.02%	0.24%	1.37x
	Ref 2	86.12%	-0.66%	3.00x
	Ref 3	80.56%	-6.22%	4.97x

As shown in Table 4.5, the optimal solutions that found by the proposed method require less number of multiplications than the standard VGG-11. At the same time, the classification accuracy of Ref1 is outperformed the original VGG-11. Ref2 requires 3x less multiplications with only 0.66% decrease in classification accuracy, compared with original VGG-11.

The Table 4.6 demonstrates models' classification accuracy which are found by the proposed method against the other peer competitors who also applied EAs to optimise the CNNs in recent years on MNIST, Fashion-MNIST and CIFAR-10 datasets. It can be observed, all reference points evolved by the three-objective optimisation method get the highest classification accuracy on MNIST dataset, compared with other EA-based optimisation methods. For the Fashion-MNIST dateset, the Ref 1 from the LeNet

Table 4.6 Comparison between the proposed method and other evolutionary optimisation methods for optimising CNN architecture.

Dataset	Model	Classification Accuracy
MNIST	B. Cheung and C. Sable [53]	98.54%
	LeNet CNN Ref 1	**99.52%**
	LeNet CNN Ref 2	**99.48%**
	LeNet Ref 3	**99.38%**
Fashion-MNIST	D. Kang and C. Ahn [93] Exp. A	92.90%
	D. Kang and C. Ahn [93] Exp. B	93.24%
	D. Kang and C. Ahn [93] Exp. C	92.98%
	EvoCNN [94]	92.72%
	LeNet Ref 1	**92.96%**
	LeNet Ref 2	**92.75%**
	LeNet Ref 3	**92.67%**
CIFAR-10	MOCNN [60] Setting 1	84.50%
	MOCNN [60] Setting 2	85.20%
	J. Prellberg and O. Kramer [57] Setting 1	85.00%
	J. Prellberg and O. Kramer [57] Setting 2	77.60%
	J. Prellberg and O. Kramer [57] Setting 3	70.40%
	LeNet Ref 1	**83.47%**
	LeNet Ref 2	**82.46%**
	LeNet Ref 3	**80.73%**
	Five-layer CNN Ref 1	**84.76%**
	Five-layer CNN Ref 2	**83.57%**
	Five-layer CNN Ref 3	**82.16%**
	Six-layer CNN Ref 1	**82.77%**
	Six-layer CNN Ref 2	**82.07%**
	Six-layer CNN Ref 3	**80.56%**
	VGG-11 Ref 1	**87.02%**
	VGG-11 Ref 2	**86.12%**
	VGG-11 Ref 3	**80.56%**

optimisation results gets the third place in classification accuracy, which only has 0.28% less than the best result and 0.02% less than the second best result in network's classification accuracy. The Ref 1 and Ref 2 from the VGG-11 optimisation results lead the first and second best classification on CIFAR-10 dataset, compared with EA-based optimisation method for CNNs. Optimising LeNet by proposed method on CIFAR-10 dataset are less accurate than other networks. The main reason is because the LeNet-5 is originally designed for recognising grey-scale handwritten digits, which is not as deep as other architectures that are designed for RGB image classification, such as VGGNet.

4.4 Summary

In this chapter, a three-objective evolutionary optimisation method for optimising the convolutional layers is introduced to reduce the size of convolutional layers and explore the design space of CNNs by combining different number and multiple size of unconventional convolution kernel which was proposed in Chapter 3. The methodology developed in this chapter is an extension of the work in Chapter 3. Here, in contrast, the number of convolution kernels in each convolutional layer is no longer the same as the benchmark. The MOEA is required to make decisions about how many convolution kernels should be used in each convolutional layer, and what size and shape of convolution kernels should be involved. The experimental results show that a trade-off space between number of multiplication in convolutional layer, classification accuracy and number of kernels usage can be achieved.

The proposed method has been successfully applied to the optimisation of various CNNs on MNIST, Fashion-MNIST and CIFAT-10 datasets. Compared with conventional designs in original benchmark architectures, the proposed method demonstrates significant improvements on saving computational resources and sometimes with increases in classification accuracy. there is also a comparison between the proposed method and

other EA-based optimisation method for CNNs. Compared with these peer competitors, the proposed method leads the best optimisation results on MNIST and CIFAR-10 datasets.

Chapter 5

8-bit Integer Quantisation through Evolutionary Optimisation

In this chapter, an evolutionary algorithm (EA) based adaptive integer quantisation method is proposed to reduce network size. The proposed method uses single objective rank-based evolutionary strategy to find the best quantisation bin boundaries for fixed quantised bit width. Due to computational cost, as a proof of concept, the performance of the proposed method is evaluated on a relatively small CNN, the LeNet-5 architecture, using the CIFAR-10 dataset. The aim is to devise a methodology that allows adaptive quantisation of both weights and bias, reducing them from 32-bit floating point to 8-bit integer representation for LeNet-5, while retaining accuracy. The experiments compare straight-forward (linear) quantisation from 32-bits to 8-bits with the proposed adaptive quantisation method. The results show that the proposed method is capable of quantising CNNs to lower bit width representation with only a slight loss in classification accuracy.

5.1 Overview

Convolutional Neural Networks (CNNs) have been widely used in image classification and object detection tasks. In recent years, many CNN architectures have been developed, such as GoogLeNet [7] and AlexNet [6]. The state-of-the-art achievements of designing CNN architectures have demonstrated significant improvements on network classification accuracy on various benchmark datasets. In order to increase the classification accuracy on specific tasks, modern CNNs are usually designed with extremely deep layers[5]. Most of the weights and bias values of the state-of-the-art CNNs are implemented and trained using 32-bit floating point representation. However, this poses significant challenges to embedded hardware systems, especially processing deep CNNs on memory-constrained platforms. Therefore, in order to implement deep CNNs in certain memory-constrained platforms, such as Field-Programmable Gate Arrays (FPGAs) and embedded devices, the memory usage of processing CNNs has to

be reduced. And also, applying integer representation can be much more efficiently processed in hardware systems.

Evolutionary algorithms (EAs) are today a state-of-the-art methodology in solving hard multi-objective optimisation problems [45]. EAs were created by taking inspiration from natural selection and survival of the fittest from Darwinian Evolution and modern genetics [16]. EAs 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 [19–21].

This chapter investigates adaptive integer quantisation of feed-forward CNN architectures for application in resource-constrained scenarios by using EAs. An EA-based adaptive integer quantisation method is proposed to quantise pre-trained CNNs, i.e. post-training quantisation [95], to 8-bit integer weights and bias. In order to quantise CNNs to smaller bit width integer representation while keeping the classification accuracy as high as possible, the proposed method uses the CNN classification accuracy as fitness during the optimisation. The proposed method is then evaluated on LeNet-5 architecture [27] with CIFAR-10 dataset [9]. As the results show, the proposed method quantises both weights and bias values across all layers for LeNet-5 from 32-bit floating point representation down to 8-bit integer value with only slight increases in classification error.

This chapter is organised as follows: Section 5.2 gives an overview of the current approaches of different quantisation methods for pre-trained CNNs, as well as different data representation for weights and bias. Section 5.3 describes the methodology that designs and implements EAs to quantise pre-trained CNNs and explains its features. Section 3.4 shows the test experiments applying the proposed method to quantise

weights and bias values for LeNet-5 with CIFAR-10 dataset and provides a proof-of-concept of the proposed method. Finally, Section 5.5 summarises the paper and discusses further works.

5.2 Related Work

In order to process CNNs more efficiently and quicker, quantisation methods aim to reduce the size of CNN models while keeping models as accurate as possible [96, 97].

5.2.1 Vector Quantisation

Vector quantisation is a method for compressing densely connected layers to make CNNs smaller. It quantises groups of similar numbers together [98]. The disadvantage is that, in order to reduce the computation and memory resources usage, vector quantisation always has some inherent loss of accuracy.

Gong et al [99] proposed a vector quantisation method that quantise weights of fully-connected layers of a CNN, as parameters in fully-connected layers take the most of memory space in CNNs [100]. Denil et al [101] applied vector quantisation to significantly reduce the number of dynamic parameters in deep models by representing the weight matrix as a low rank product of two smaller matrices. Other approaches use bio-inspired generative models to encode large numbers of parameters and network structures in compact representations [102, 103]. Jiang Su [100] conducted research on reducing network redundancy of artificial neural networks implemented on FPGAs. Their work suggests that the network redundancy can be reduced at the data-level and model level. The hardware system after reducing network redundancy shows that both logic resources and on-chip memory usage can be reduced.

5.2.2 Low-precision Representation for Neural Networks

Fixed-point quantisation is normally used in improving CNNs performance. In earlier works, low precision fixed-point representation has been used in training CNNs [104]. Lin et al [105] investigated a method to identify the optimal fixed-point bit width allocation across convolutional layers based on the number of parameters in each convolutional layer. Wu et al [106] proposed a quantisation scheme for both convolutional layers and fully-connected layers that accelerates the CNNs processing speed and reduce the memory costs for processing CNNs in mobile devices. An FPGA-based low precision fixed-point implementation has been investigated by Sit et al [107].

In recent years, a more efficient low precision neural network model has been proposed, called Binarized Neural Networks (BNNs) [108]. In the BNNs, all of weights are approximated with binary ones. The core idea is to binaries the weight values in the weight matrices and the function values of each activation function down to -1 and 1 at the same time. Due to the extremely low precision of BNNs, it shows more significant performance gain on FPGA implementation. M. Rastegari et al. [97] proposed a XNOR-Net where both of the weights and inputs are binary values. So that, compared with BNNs, the XNOR-Net can further improve the memory saving. Their experimental results show that the XNOR-Net can reduce the memory usage by 32x and 58x speedup compared with full-precision AlexNet [6] for ImageNet dataset [10] on GPU and CPU implementation. Their results also show that the classification accuracy with Binary-Weight-Network version of AlexNet is the same as the full-precision AlexNet. Then, FINN [109] and FP-BNN [110] provides the solution for mapping BNNs onto FPGA. Their results show that because of less memory and computing resource usage, the FPGA implementation of BNNs get the highest throughput at the time of publication as well as less power consumption.

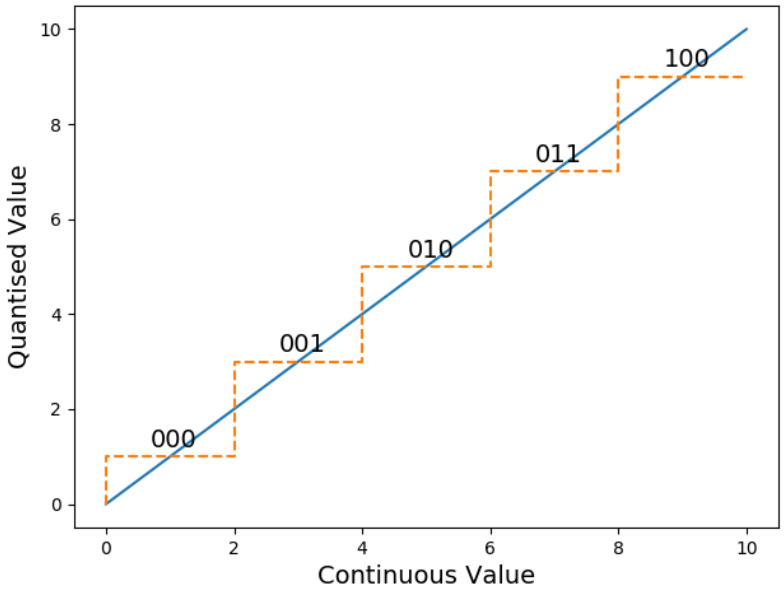
5.3 Methodology

The main idea in this work for quantising pre-trained CNNs to low-precision integer representation is to consider how the width of each quantisation bin can be optimised while keeping the classification accuracy of the model as high as possible. In this section, a rank-based evolutionary strategy for finding the best quantisation bin widths after converting 32-bit to 8-bit representation is introduced. Choosing an integer-based representation brings additional benefits when implementing the model in hardware, e.g, FPGAs or embedded systems. There are other potential benefits when combining the proposed adaptive quantisation concept with that of approximate computing hardware. For example, we have explored the use of an efficient approximate multiplier in previous work [1] that may be a good candidate for exploring this in the future.

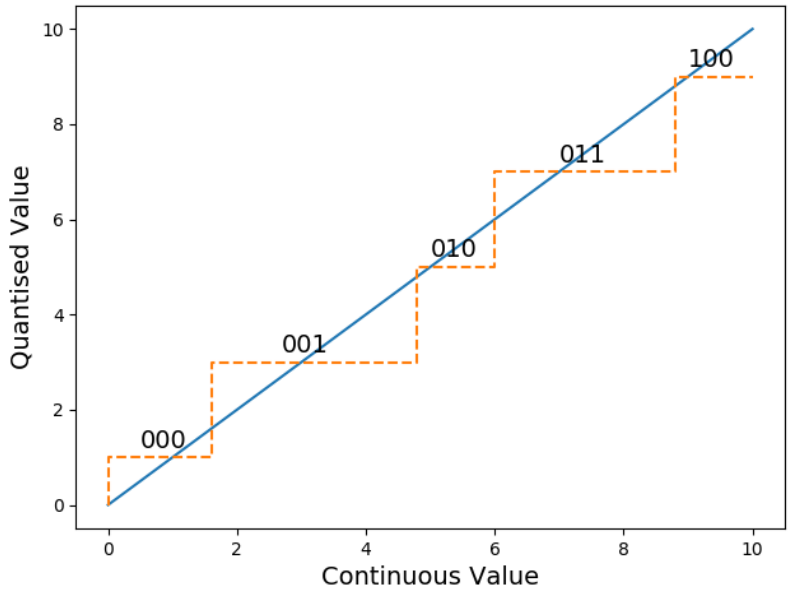
5.3.1 Integer Quantisation

Quantisation methods are normally used in signal processing, where an analogue signal needs to be converted into the digital domain. During quantisation, a range of continuous values is quantised into a single numerical value. For example, 8-bit integer quantisation of continuous real values between 0 and 1 divides the range [0..1] into 256 equally-sized intervals and maps each of them to one of the corresponding 256 integer values which represent the original set. The key of implementing a quantisation method is to decide the quantisation boundary that determines what kinds of values can be quantised to a representative value, as shown in Fig. 5.1a.

Following the quantisation method from above, it is hard to determine what sizes of the quantisation bins can make the classification accuracy of quantised weights and bias as high as possible for a given pre-trained CNN. The adaptive integer quantisation method proposes to use an evolutionary algorithm to find upper and lower boundaries



(a)



(b)

Figure 5.1 (a) An example of linear quantisation. The blue line shows 1000 evenly spaced numbers from 0 to 10. The orange line represents that quantise the red line value to 5 representative values. (b) An example of adaptive (non-linear) quantisation. The blue line shows 1000 evenly spaced numbers from 0 to 10. The orange line represents that quantise the red line value to 5 representative values with different boundaries.

for each quantisation bin within the low-precision integer representation, as shown in Fig. 5.1b.

5.3.2 Evolutionary Approach for Adaptive Integer Quantisation

In order to explore the bin boundaries, a rank-based evolutionary strategy to find out the best bin location and size for quantising CNNs is developed. An overview of the proposed adaptive integer quantisation method is shown in Fig. 5.2.

For the EA, the size of each individual will depend on the quantisation bit width and the depth of the CNN. If the quantisation bit width is n, the number of bins under this bit width will be 2^n. Then, the size of the individual for a given CNN with m layers will be $(2^n - 1) \times m \times 2$, including both weights and biases for all layers. For example, if the adaptive integer quantisation is used to quantise a two-layer network with 4-bit integer representation, each individual will contain 60 genes. Each gene is initialised with a random number from 0 to 1. This number represents where the range interval is cut for each quantisation bin. The pre-trained model contains weights and biases with 0 to 1 32-bit floating point value, therefore, the genome will cut the floating value and generate related quantisation bins. An example of 3-bit quantisation genome has been shown in Fig. 5.3.

After the initial population is generated, mutation operation is used to generate the offspring of the initial population. The mutation operation modifies the gene between the range of 0 and 1 based on a given mutation probability. The mutation operation is described in Algorithm 5.

Then, both the initial population and the offspring population are decoded to representative quantisation bins. The fitness of the individuals is evaluated by running

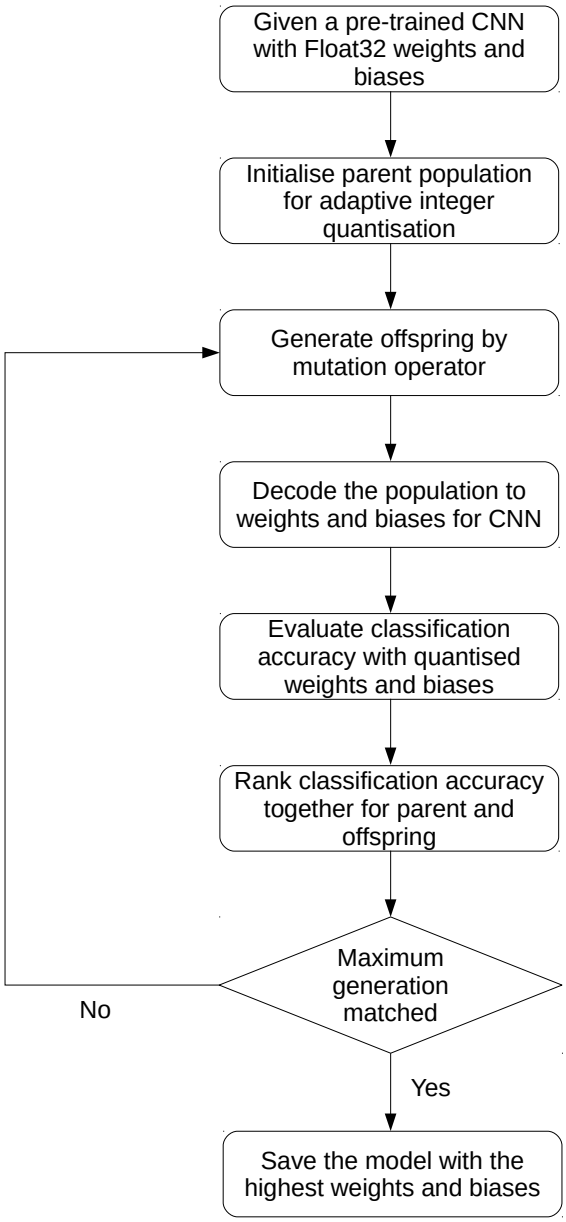


Figure 5.2 Overview of the adaptive integer quantisation loop. The method generates a set of bins to quantise the original weights and biases. The classification accuracy of each individual is evaluated on a test dataset, and represents as fitness to the EA. Then, the classification accuracy of each individual is ranked by the EA. The individual with the highest classification accuracy will be used as the parent population to the next generation.

classification accuracy of the CNNs inference using the quantised integer weights and biases on the test dataset. Then, the classification accuracy of both parent and offspring populations are ranked from the highest to lowest. The individual with the highest

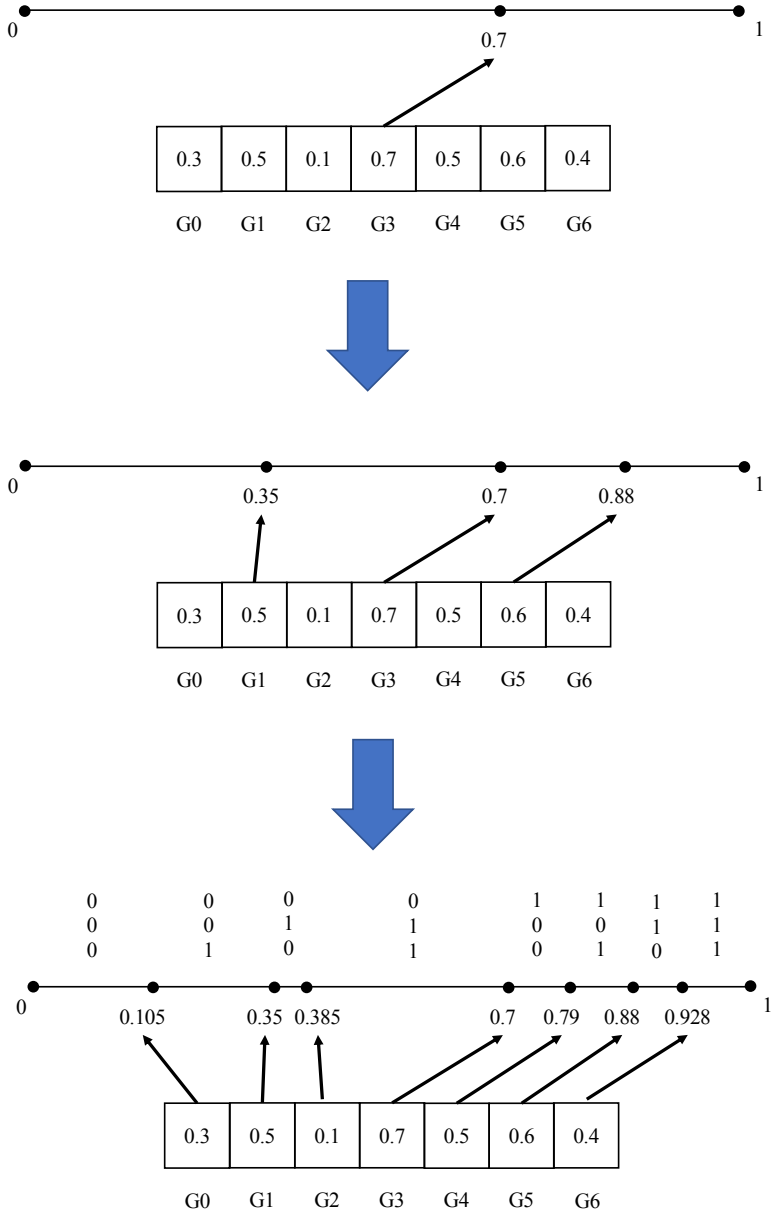


Figure 5.3 The figure shows how to decode 3-bit quantisation bin widths from the parameters stored in the genome. The genome is encoded in the same order as the bin from minimum to maximum value (left to right). The middle gene (G3) between G0 and G6 divides the range from minimum value (0) to maximum value (1) according to its parameter value. Then, the middle gene (G1) between minimum value (0) and gene G3 divides the value from minimum to G1's parameter value. The middle gene (G5) between the maximum value (1) and gene G3 cut the value from G5's representative value to maximum value and so on. This iterative bisection method is used to simplify mutation operation and avoid convergence towards large clusters of small bins.

Algorithm 5 Mutation operation

Procedure: Mutation (P, ρ). ▷ Population (P) has N individuals and M genes in each individual. The mutation probability is ρ.

```
Offspring population (O) ← ∅
for i ← 1 to N do
    for j ← 1 to N do
        if random(0, 1) < ρ then
            O_{i,j} ← random[0, 1]
        else
            O_{i,j} ← P_{0,j}
        end if
    end for
end for
```

rank will be taken as the parent genome and then mutated to obtain the population for the next generation. The loop will be running until the specified number of generations is reached.

The overall system of adaptive integer quantisation shows in Algorithm 6 and works as follows:

1. Get weights and biases from pre-trained CNN model.
2. Generate genomes and decode genomes to respective weights and biases.
3. Evaluate classification accuracy of CNN inference by quantised weights and biases.
4. Rank results from CNN inference.
5. Repeat 2) to 4) until the system reaches the maximum generation
6. The model with the highest classification accuracy is saved as quantised model of adaptive integer quantisation.

Algorithm 6 Evolutionary Strategy for adaptive integer quantisation

Procedure: Evolutionary Strategy (M, N, $f(x)$). ▷ evolving N individuals for M generation with fitness function of $f(x)$, $\forall x \in X$, where $f(x) = \max[classification\ accuracy]$ and $X \subset [0, 1]$

1: Initialise parent population $P_1 = [x_1,\ x_2,\ ...,\ x_N]$
2: Offspring population $O_1 \leftarrow$ Mutation (P_1)
3: **for** $i \leftarrow 1$ *to* M **do**
4: $C_i \leftarrow O_i \cup P_i$ in size $2N$
5: $R_i \leftarrow f(C_i)$ ▷ evaluating each individual in C_i with fitness function $f(x)$
6: $R_i \leftarrow$ Descend-Sort(R_i) ▷ ranking the individual with the fitness (classification accuracy) from high to low in C_i
7: $C_i \leftarrow$ Ranking C_i based on R_i
8: $P_{i+1} \leftarrow \emptyset$
9: **for** $j \leftarrow 1$ *to* N **do**
10: $P_{i+1,j} \leftarrow C_{i,j}$
11: **end for**
12: $O_{i+1} \leftarrow$ Mutation(P_{i+1})
13: **end for**

5.4 Experimental Results

In this section, the proposed method has been tested and evaluated using the LeNet-5 [27] benchmark CNN architecture. A small CNN, the Lenet-5 architecture, is used here as an initial benchmark architecture to illustrate the performance achieved by our proposed adaptive quantisation method.

5.4.1 Experiment Setup

The original Lenet-5 architecture is built using two convolutional layers, pooling layers connected at the end of each convolutional layer, fully-connected layers and a classification at the end. In order to improve the classification accuracy of the benchmark CNN, the number of convolution kernels is increased in each of the convolutional layers and the number of nodes in fully-connected layers is also increased. All convolutional

layers and fully-connected layers are followed by rectified linear unit (ReLU) [111] as activation function. The overall architecture is shown in Fig. 5.4.

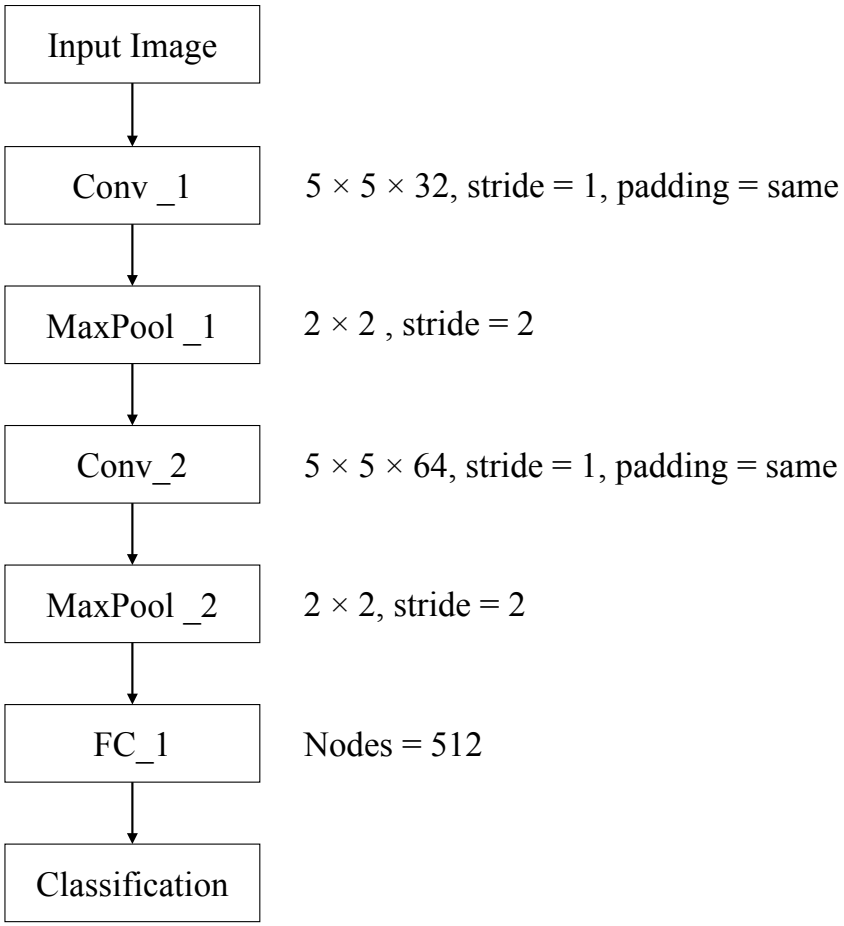


Figure 5.4 The benchmark architecture that is used to test the proposed method. The network consists of two 2D convolutional layers, which involve 32 and 64 convolution kernels respectively. All of the kernels are sized 5x5 and the stride is 1. Each convolutional layer is followed by a max-pooling layer which is sized 2x2 and a stride of 2. There is a fully-connected layer featuring 512 nodes connected at the end of the second max-pooling layer. Finally, a classification layer is used to predict the best classification label to best describe the image.

The proposed method is applied to CIFAR-10 dataset [9], which in total contains 60,000 images. The CIFAR-10 is split into a training set of 50,000 images and a test set of 10,000 images. Each image is a 32×32 RGB image, associated with a label from 10 classes. The Lenet-5 is trained by stochastic gradient descent (SGD) method with a

batch size of 64. The weights are initialised using a truncated normal distribution with standard deviation of 0.1. The Adam optimiser [90] is used with an initial learning rate of 0.001. The 'softmax cross-entropy loss' is used as the loss function. The CNN has been trained for 50 epochs in total and the learning is decreased to 1×10^{-4} after 10 epochs. After 50 epochs of training, the benchmark CNN has reached the classification accuracy of 78.28%.

In order to explore the best quantisation bin distribution, the proposed method is built on a single-objective, rank-based evolutionary strategy. The initial population is generated randomly from 0 to 1. The experiment is set to run for 200 generations using a population size of 100. Only mutation operation is applied with the mutation rate set to a probability of 0.1.

5.4.2 Results for Accuracy

The results of the proposed method are reported in Fig. 5.5. The experiment is setup to compare the classification accuracy between the original 32-bit floating point (Float32), linear 8-bit integer (INT8) quantisation and adaptive 8-bit integer quantisation. The original Float32 values are the ones from the pre-trained LeNet-5 which has weight distributions between -1 and +1 in Float32 representation. The linear INT8 quantisation represents linearly quantised weights from the pre-trained Float32 network that will generate a weights distribution between -128 to +127 as a baseline comparison. The linear INT8 quantisation method is used in both convolutional layers, but the Float32 representation is kept for the fully-connected layers. The simple linear INT8 quantisation reaches a reduced classification accuracy of 64.79%.

It can been seen from Fig. 5.5 that the classification accuracy of quantised networks is increased significantly during the first 40 generations and continues to sightly increase after the 50th generation. The final best fitness after 200 generations has

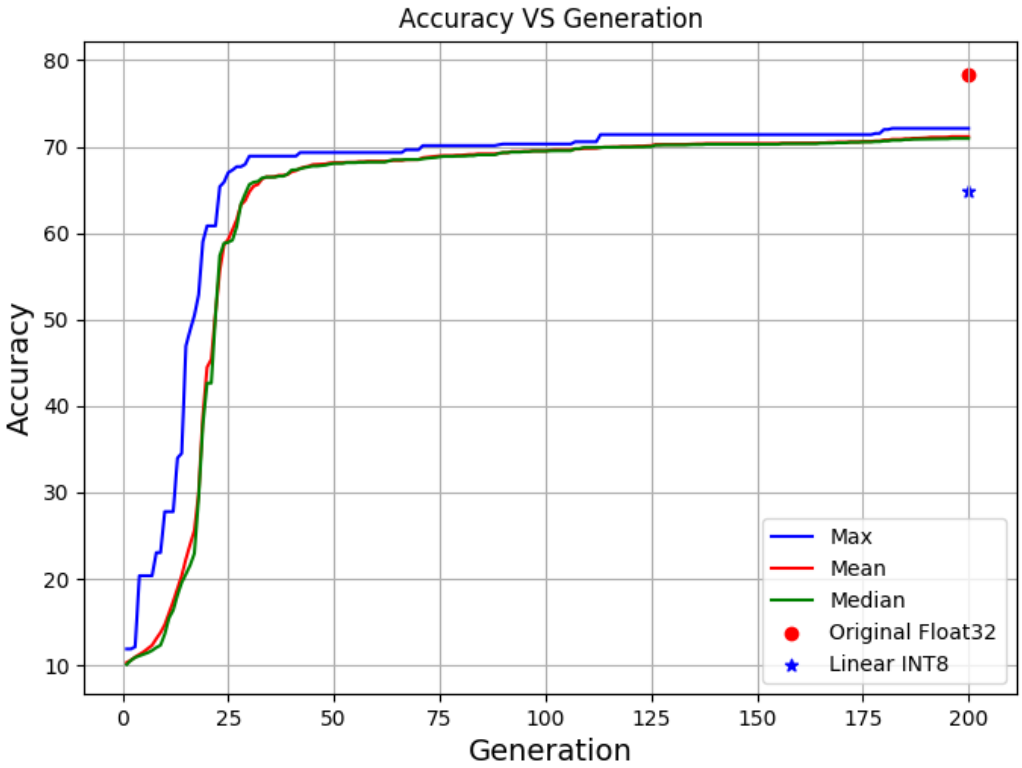
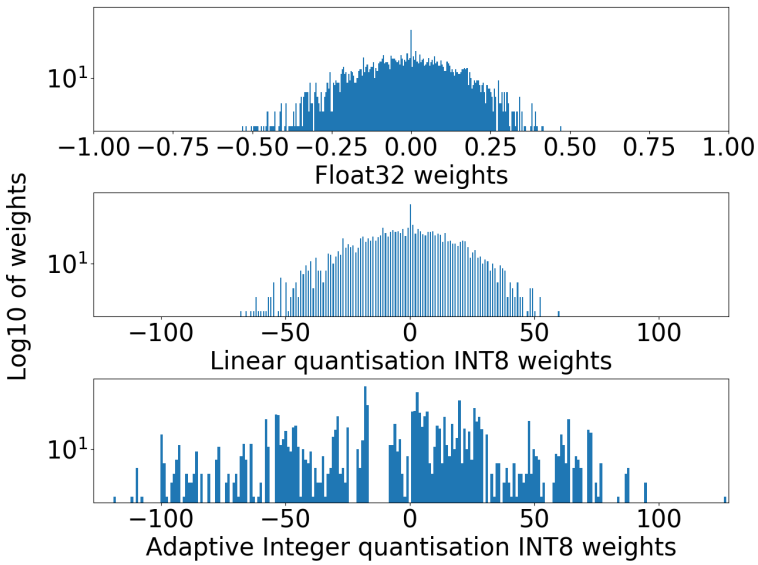


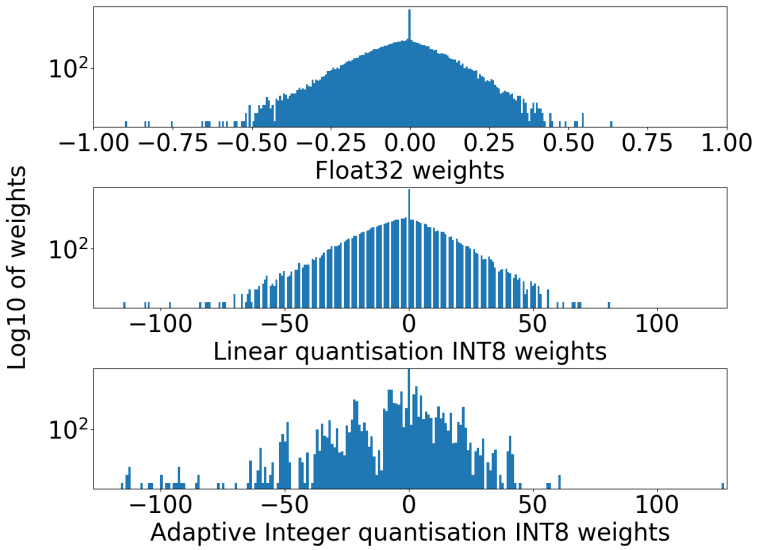
Figure 5.5 Classification Accuracy vs. Generation. The blue line shows the best fitness of each generation. The maximum, minimum and mean classification accuracy of quantised model are reported to show the performance of the proposed method. The read and green lines show the mean and median of each generation, respectively. The red dot indicates the classification accuracy of the pre-trained LeNet-5 on CIFAR-10 dataset using the original 32-bit floating point representation, and the blue star represents the classification accuracy of LeNet-5 with linear 8-bit integer quantised weights.

achieved classification accuracy of about 73%. Although the classification accuracy of the adaptive integer quantised network is lower than that from the original Float32 representation, it features significantly better accuracy than the network using basic linear INT8 quantisation.

Fig. 5.6 shows the distribution of quantised weights for convolutional layers of LeNet-5 after 100 generations. As can be seen from Fig. 5.6a, the original weights distribution of the pre-trained LeNet-5 is similar to a normal distribution ranging between -0.6 to +0.5. Accordingly, the linear 8-bit integer quantisation only covers the distribution range between -80 to +70, without changing the weights distribution shape. However,



(a)



(b)

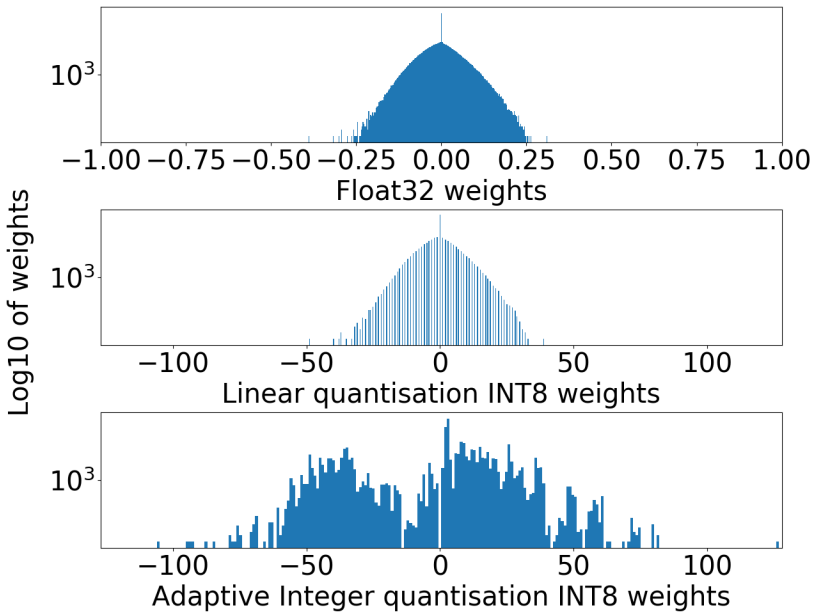
Figure 5.6 The comparison between original weights distribution, 8-bit linear integer quantised weights and 8-bit adaptive integer quantised weights for convolutional layers of test CNN. The y-axes are log10 scale so that the profile of the distribution becomes more visible. (a) Weights distributions of pre-trained Float32 weights, linear quantised INT8 and adaptive integer quantised INT8 for first convolutional layer. (b) Weights distributions of pre-trained Float32 weights, linear quantised INT8 and adaptive integer quantised INT8 for second convolutional layer.

as the figure shows, the 8-bit adaptive integer quantisation does not only expand the weights distribution to the full range of -128 to +127, but it also changes the original weights distribution's shape profile.

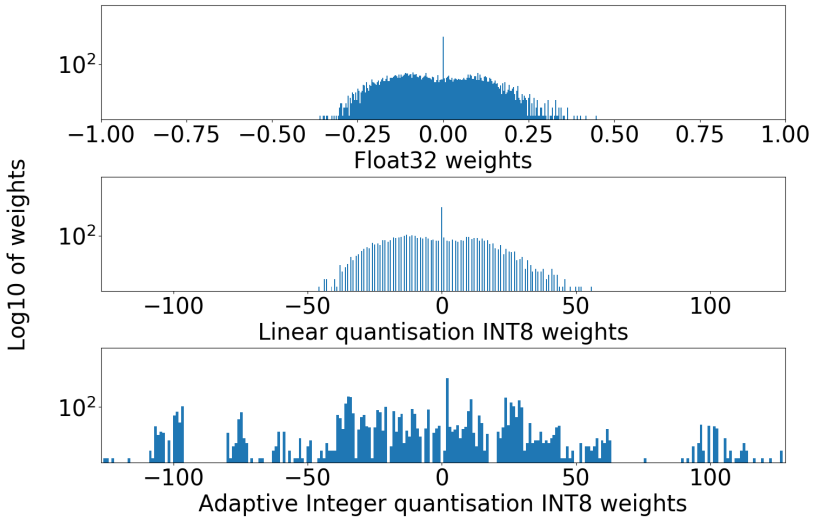
After applying the adaptive integer quantisation method to the pre-trained model, the original normal distribution's profile has been significantly changed, and also the range of the adaptive integer quantisation distribution is much larger than that from the linear integer quantisation. Therefore, under the same quantised bit width, adaptive integer quantisation can more efficiently express information than simple linear integer quantisation. The comparison of weights distribution between pre-trained Float32, linear quantised INT8 and 8-bit adaptive integer quantisation for the fully-connected layer and classification layer are shown in Fig. 5.7.

5.5 Summary

The proposed adaptive quantisation method applies a rank-based evolutionary algorithm to generate different bin locations and sizes for quantising a pre-trained CNN. As the experimental results show, the proposed method can successfully quantise pre-trained CNNs with reasonable classification accuracy. Although, after running 200 loops, the quantised CNN shows same loss in classification accuracy compared with the original 32-bit floating point representation, however, the size of the quantised model is much smaller than the original. In this case, the memory used to store the weights and biases values are reduce 4 times, as all weights and biases values in the network are converted from 32 bits down to 8 bits. The results show that the proposed method provides a significant improvement over the results obtained from simple linear integer quantisation.



(a)



(b)

Figure 5.7 Comparison between original weight distribution, 8-bit linear integer quantised weights and 8-bit adaptive integer quantised weights for fully-connected CNN layers. The y-axes are log10 scale so that the profile of the distribution becomes more visible. (a) Weights distributions of pre-trained Float32 weights, linear quantised INT8 and adaptive integer quantised INT8 for the first fully-connected layer. (b) Weights distributions of pre-trained Float32 weights, linear quantisaed INT8 and adaptive integer quantised INT8 for classification layer.

Chapter 6

Adaptive Integer Quantisation for Various Bit-Width Configurations

6.1 Overview

In the previous chapter, an EA-based adaptive quantisation method has been proposed to quantise the weights and biases of pre-trained CNNs from 32-bit floating point to 8-bit integer representation. As a result, the parameter size of target CNNs has reduced to 4x smaller than using 32-bit floating point, and also, the classification accuracy of the quantised model by EA-based adaptive quantisation method is significantly higher than using simple integer quantisation. In this chapter, applying mixed-precision number representations to quantise CNNs is investigated.

The investigation of different bit-width representation of EA-based adaptive integer quantisation is divided into two categories. Firstly, in order to find out how different integer representations between convolutional layers and fully-connected layers effect a network's classification accuracy, and how different bit-width settings between convolutional layers and fully-connected layers can be applied in the proposed quantisation method. Second, an investigation of how different integer representations between weights and biases affect the network's classification accuracy is carried out. To do this, the adaptive integer quantisation method is configured to quantise the weights and biases of a pre-trained network with mixed-precision number representation.

Furthermore, the quantisation methodology developed here is combined with the computational cost optimisations proposed in Chapter 3 and Chapter 4.

The rest of this chapter is organised as follows: Section 6.2 introduces the configuration of EA-based adaptive integer quantisation method, which is used to carry out the two investigations from above, i.e. the configurations of adaptive integer quantisation. Section 6.3 demonstrates some experimental results of applying different bit-width to quantisation pre-traind CNNs by adaptive integer quantisation, and also the proposed methodology is applied to quantise the models which are found in Chapter 3 and Chapter 4. Finally, Section 6.4 concludes this chapter.

6.2 Configuration of the Evolutionary Algorithm

6.2.1 Mixed-Precision between Convolutional Layers and Fully-Connected Layers

For the first investigation, i.e applying different integer representation for convolutional layers and fully-connected layer, the population is configured as follows.

$$C_{size} = [N_{conv} \times (2^{B_{conv}} - 1) + N_{fc} \times (2^{B_{fc}} - 1)] \times 2 \tag{6.1}$$

The equation 6.1 describes the number of genes in each individual which is applied to the EA-based adaptive integer quantisation, where C_{size} is the total number of genes in each individual, N_{conv} and N_{fc} represent the number of convolutional layer and the number of fully-connected layer in target CNN, B_{conv} indicates the quantisation bit-width for convolutional layers and B_{fc} represents the quantisation bit-width for fully-connected layers. For each layer, both weights and biases are targeted to be quantised by adaptive integer quantisation at the same time, therefore, chromosomes include both weights and biases in the EA representation. For example, to quantise with 8-bit integer representation in convolutional layers and 4-bit integer representation in fully-connected layers for a two convolutional layer and two fully-connected layers CNN, the number of genes of each individual will be 1080.

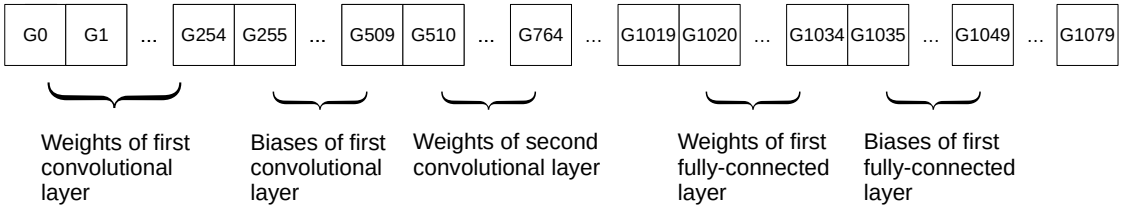


Figure 6.1 An example of the chromosome that is used for 8-bit integer in convolutional layers and 4-bit integer in fully-connected layers. Each set of genes are then decoded by the method described in Fig 5.3 in Chapter 5.

The Fig 6.1 shows an example of a chromosome that is used for quantising the CNN for 8-bit integer in convolutional layers and 4-bit integer in fully-connected layers by the EA-based adaptive integer quantisation. The chromosomes are encoded as the same order as the layers in the target CNN's architecture. As the example shows in Fig 6.1, quantising a CNN with two convolutional layers and two fully connected layer by 8-bit integer representation in convolutional layers and 4-bit integer representation in fully-connected layers, the first 255 genes indicates the bin boundaries for quantising the weights of the first convolutional and the second 255 genes represents the bin boundaries for quantising the weights of the first convolutional layers. Then the third 255 genes and fourth 255 genes indicate the bin boundaries for quantising weights and biases of the second convolutional layer, respectively. Finally, fifth 15 genes, sixth 15 genes, seventh 15 genes and eighth 15 genes indicate the bin boundaries for quantising weights and biases of the first fully-connected layers and weights and biases of the second fully-connected, respectively.

6.2.2 Mixed-Precision between Weights and Biases

Second investigation is carried on finding out whether is possible to apply different bit-width integer representation for weights and bias on target CNN.

$$C_{size} = (N_{conv} + N_{fc}) * [(2^{B_{weight}} - 1) + (2^{B_{bias}} - 1)] \tag{6.2}$$

Equation 6.2 is used to calculate the chromosome size that will be used for quantising a pre-trained CNN with different integer representations for the EA-based adaptive integer quantisation. Similarly, N_{conv} and N_{fc} indicate the number of convolutional layers and the number of fully-connected layers of the target CNN, respectively. B_{weight} represents the quantisation bit-width for weights in each layer and B_{bias} represents the quantisation bit-width for biases in each layer.

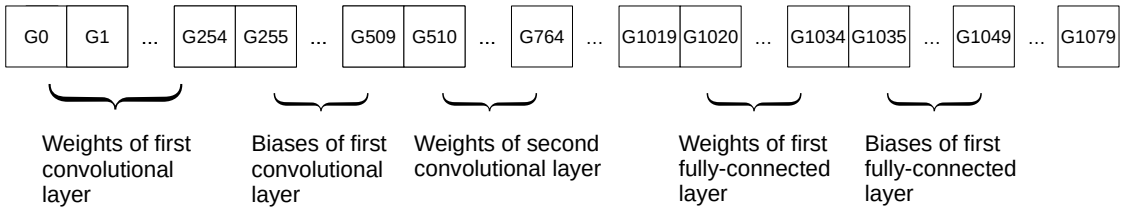


Figure 6.2 An example of the chromosome that quantising a pre-trained CNN with two convolutional layers and two fully-connected layers by 8-bit integer in weights and 4-bit integer in biases. Then, each set of the genes is decoded by the method described in Fig 5.3 in Chapter 5.

The Fig 6.2 illustrates an example of a chromosome which describing the EA representation for quantising pre-trained CNN by EA-based the adaptive integer quantisation with 8-bit integer for weights and 4-bit integer for biases. Similar to the first investigation, the chromosome is encoded in the same order as the pre-trained CNNs. For instance, in Fig 6.2, the 255 genes represent the quantisation bin boundaries of the weights in first convolutional layer and followed by 15 genes that indicate the quantisation bin boundaries of the bias in first convolutional layers. From 510th to 764th and from 765th to 780th genes are encoded to represent the weights and biases of the second convolutional layers, respectively. There are totally 1080 genes used for quantising a network with two convolutional layers and two fully-connected layers using 8-bit weights and 4-bit biases.

6.3 Experimental Setup and Results

6.3.1 Experimental Setup

In this section, the CNN used for testing the EA-based adaptive quantisation method is the same as in Chapter 5, featuring two convolutional layers with 32 and 64 5×5 convolution kernels, respectively. Two max pooling layers are connected at the end of each convolutional layer, and a fully-connected layer with 512 neurons and a

classification layer is connected at the end. The activation function used in this CNN is rectified linear unit (ReLU) [111]. Fig 6.3 illustrates the test CNN architecture.

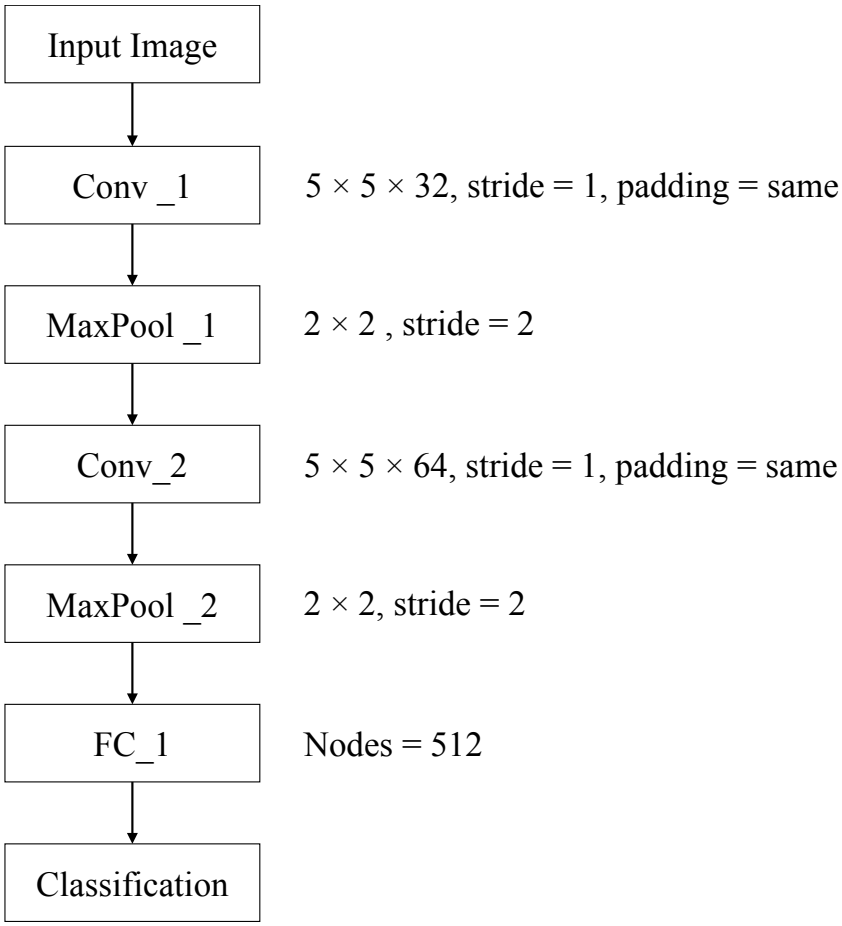


Figure 6.3 The CNN architecture is used to test the EA-based adaptive integer quantisation.

The CNN is trained on CIFAR-10 dataset [9] by SGD method with a batch size of 64. Weights are initialised by a truncated normal distribution with a standard deviation of 0.1. The Adam optimiser [90] is used with an initial learning rate of 0.001. The softmax cross-entropy loss is used as the loss function. The CNN has been trained for 50 epochs in total and the learning rate is decreased to 1×10^{-4} after 10 epochs. After 50 epochs of training, the benchmark CNN has reached a classification accuracy of 78.28%.

The EA-based adaptive integer quantisation loop is set to run for 200 generations with a population size of 10 individuals. Only mutation operation is applied with the mutation rate set to a probability of 0.1.

6.3.2 Experimental Results

In the first experiment, the weights and biases in convolutional layers are quantised to 8-bit integer and the weights and biases in fully-connected are variable. In this case, the best fitness obtained from 8-bit adaptive integer quantisation in Chapter 5 is used to initialise the population, thus, the first individual in the initial population uses the quantisation bin parameters of the convolutional layers of the best results from experiments results in Chapter 5. The remaining individuals in the initial population are randomly initialised.

Seven experiments are conducted with 8-bits in convolutional layers, and 7-bits in fully-connected layers(C8F7), 8-bits in convolutional layers with 6-bits in fully-connected layers(C8F6), 8-bits in convolutional layers with 5-bits in fully-connected layers(C8F5), 8-bits in convolutional layers with 4-bits in fully-connected layers(C8F4), 8-bits in convolutional layers with 3-bits in fully-connected layers(C8F3), 8-bits in convolutional layers with 2-bits in fully-connected layers(C8F2) and 8-bits in convolutional layers with 1-bit in fully-connected layers(C8F1) . The experimental results are reported in Fig. 6.4.

As shown in Fig. 6.4, apart from C8F8, all results from EA-based adaptive integer quantisation with different bit-width combination between the convolutional layers and fully-connected layer outperform the classification accuracy of linear 8-bit integer quantisation. Compared with 8-bit adaptive integer quantisation which has been demonstrated in Chapter 5, marked as C8F8, the experimental results of C8F7, C8F4 and C8F3 have slightly higher classification accuracy than using 8-bit integer only

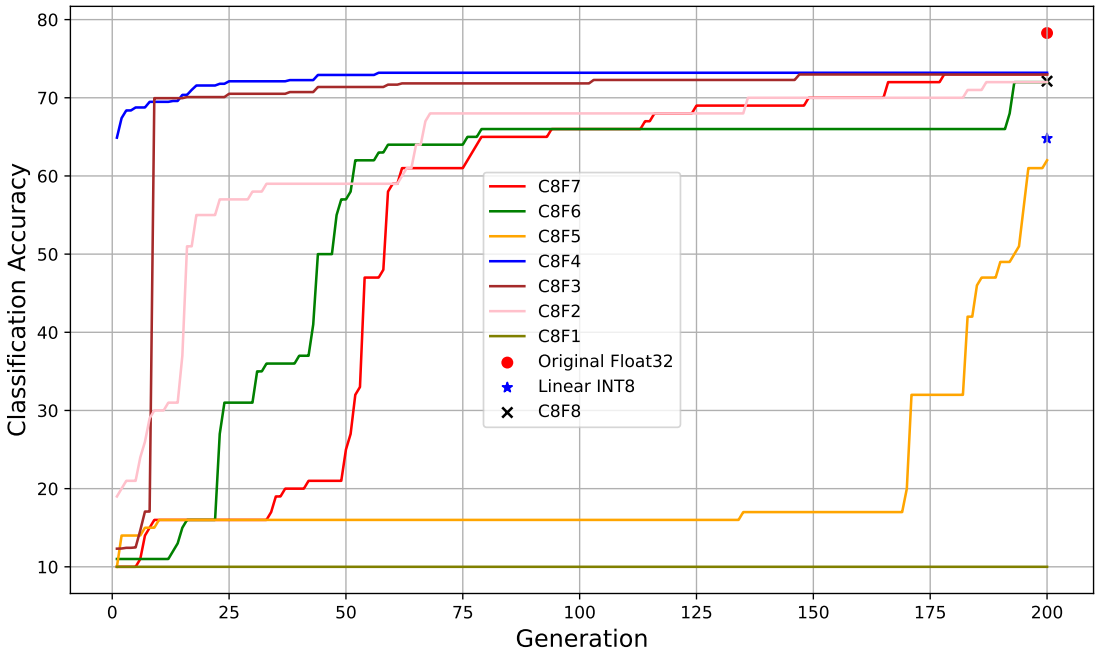


Figure 6.4 Classification Accuracy vs. Generation. In this experiment, the data precision of convolutional layers is kept as 8-bit integer representation and the data precision of fully-connected layers are reduced from 7-bit to 1-bit integer representation.

and C8F4 achieve 73.20% in classification accuracy after quantisation which is the highest classification accuracy in this set of experiments. Different from others, after 200 generations, C8F3, C8F2 and C8F1 has lower classification accuracy than the linear 8-bit quantisation. Since the classification accuracy the C8F5 starts dramatically increasing after 170th generation, while others' classification accuracy grow faster in early generations, classification accuracy of C8F5 may keep increasing after 200 generation. However, in order to make fair comparison, all experiments were set to run for 200 generation. Even reducing the precision in fully-connected layer down to 2 bits, the network's classification accuracy is still close to 8-bit representation, which only has 0.12% of classification accuracy drop after quantisation. However, the network's classification accuracy of C8F1 is always 10% and does not have any improvement during the quantisation process. This suggests that it is necessary to keep at least 2-bits in the fully-connection layer when implementing the proposed method. Table. 6.1

compares the classification accuracy of different precision after EA-based adaptive integer quantisation.

Table 6.1 Comparison between the original network, 8-bit EA-based adaptive integer quantisation, linear 8-bit integer quantisation and different bit-width in fully-connected layers.

Dataset	Model	Classification Accuracy
CIFAR-10	Original	78.28%
	Linear 8-bit	64.79%
	Adaptive 8-Bit (C8F8)	72.12%
	C8F7	73.00%
	C8F6	72.00%
	C8F5	62.00%
	C8F4	73.20%
	C8F3	72.98%
	C8F2	72.00%
	C8F1	10.00%

For the second set of experiments, the weights and biases in the fully-connected layer are quantised to 8-bit and the representation in the convolutional layers is variable. The best parameters from Chapter 5 are loaded into the first individual of the initial population. The remaining individuals in the initial population are randomly initialised. The experimental results are reported in Fig. 6.5.

It can be seen from Fig. 6.5, that all of the networks' classification accuracy of test experiments are much lower than the classification accuracy of the 8-bit EA-based adaptive integer quantisation, which is marked as C8F8. The experimental results show that even only reducing one bit of precision of weights and biases of convolutional layers, the classification accuracy has a dramatic drop, compared with applying 8-bits in all layers. The experimental results suggest that, as the data precision for convolutional layers goes down, the classification accuracy of the network decreases. Table. 6.2

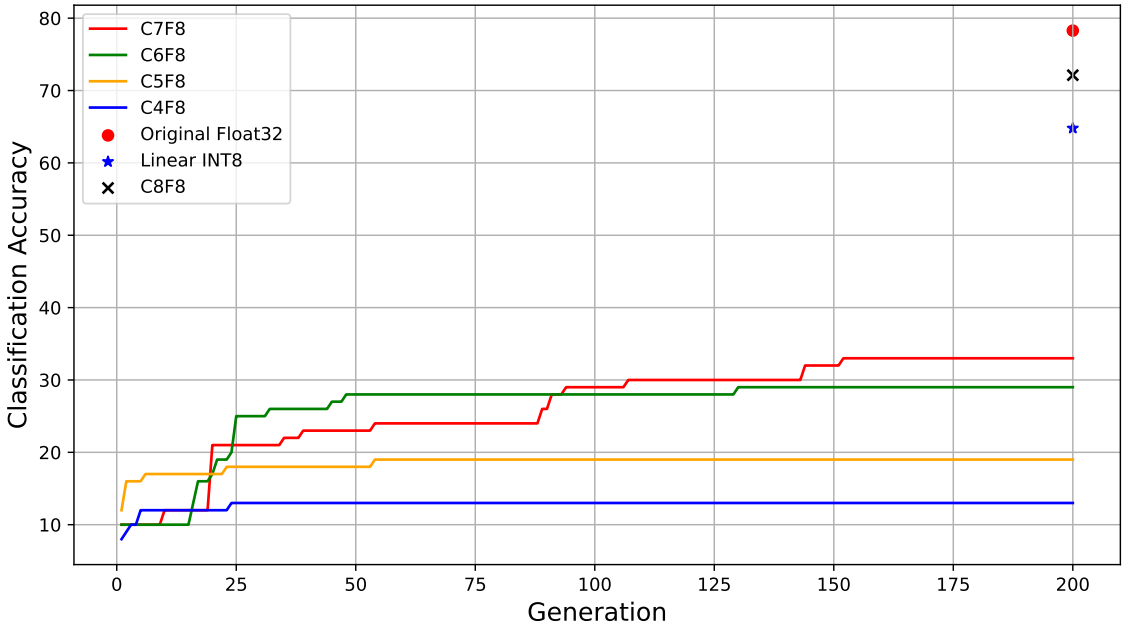


Figure 6.5 Classification Accuracy vs. Generation. In this experiment, the data precision of fully-connected layers is kept as 8-bit integer representation and the data precision of convolutional layers are reduced from 7-bit to 1-bit integer representation.

Table 6.2 Comparison between the original network, 8-bit EA-based adaptive integer quantisation, linear 8-bit integer quantisation and different bit-width in convolutional layers.

Dataset	Model	Classification Accuracy
CIFAR-10	Original	78.28%
	Linear 8-bit	64.79%
	Adaptive 8-Bit (C8F8)	72.12%
	C7F8	33.00%
	C6F8	30.00%
	C5F8	19.00%
	C4F8	12.00%

shows the comparison of classification accuracy between 8-bit fully-connected layer and different data precision of the convolutional layers.

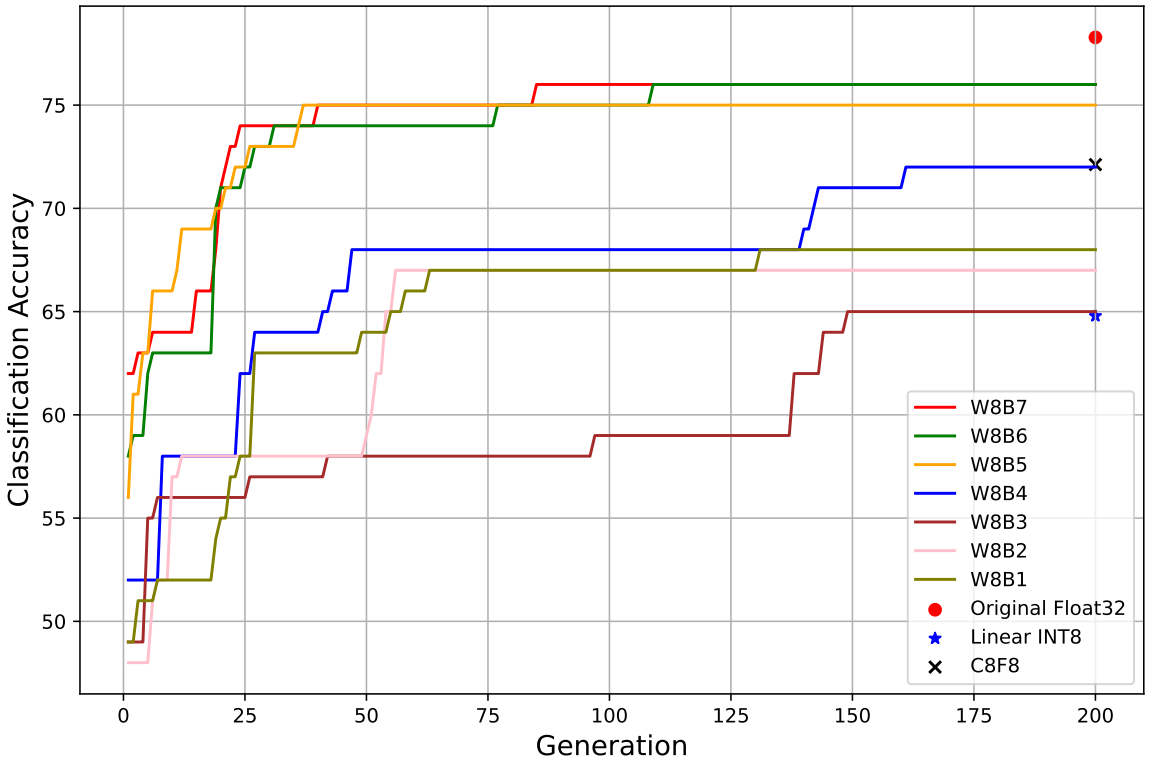


Figure 6.6 Classification Accuracy vs. Generation. In this experiment, the data precision of weights is kept as 8-bit integer representation and the data precision of biases is reduced from 7-bit to 1-bit integer representation.

In the third set of experiments, the data precision of weights in all layers is kept as 8-bit integer representation and varied the data precision for biases. In this case, the genomes of quantised weights from experimental result in Chapter 5 are again used to initialise are individual in the initial population. More specifically, the genes which represent the weights of CNN in first individual of initial population is the experimental results that evolved from 8-bit EA-based adaptive integer quantisation, and the genes of bias are initialised randomly. Other individuals in initial population are randomly initialised with numbers between 0 and 1. Seven experiments are conducted to test the mixed-precision between weights and biases. Parameter settings range from 8-bit weights with 7-bit biases (W8B7) to 8-bit weights with 1-bit biases (W8B1). The experimental results are shown in Fig 6.6.

As can been seen from Fig 6.6, in general, as the the bit-width of biases goes down, model's classification accuracy decreases. Three combinations are outperformed the classification accuracy of the basic quantised 8-bit model, which are W8B7, W8B6 and W8B5.

In the final experiment, 8-bit biases are kept, and the precision of weights is varied. The same as in previous population initialisation, genomes of quantised biases from experimental result in Chapter 5 are used as one of the individuals in the initial population and remaining individuals are initialised randomly. The experimental results are illustrated in Fig 6.7.

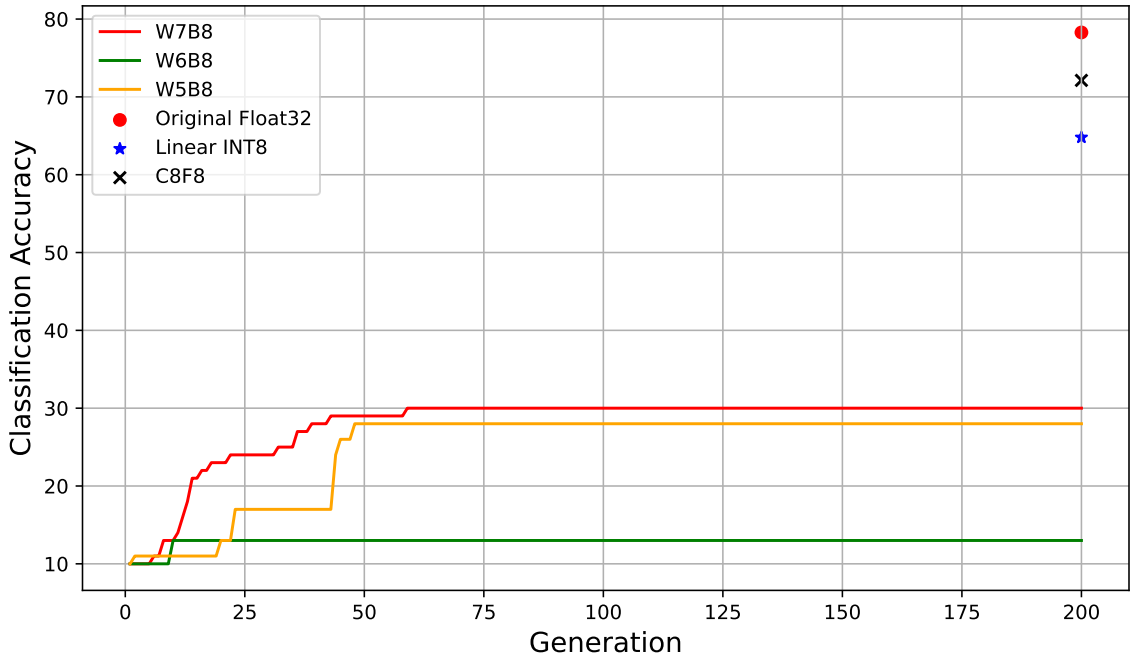


Figure 6.7 Classification Accuracy vs. Generation. In this experiment, the data precision of biases is kept as 8-bit integer representation and the data precision of weights is reduced from 7-bit to 5-bit integer representation.

As is illustrated in Fig 6.7, there is a dramatic drop in model's classification accuracy after quantising the pre-trained model with low-precision in weights. Overall, applying small bit-width integer representation for weights and 8-bit integer representation for biases leads to poor classification accuracy. Table. 6.3 shows the model's classification

Table 6.3 Comparison between the original network, 8-bit EA-based adaptive integer quantisation, linear 8-bit integer quantisation and mix-precision between weights and biases

Dataset	Model	Classification Accuracy
CIFAR-10	Original	78.28%
	Linear 8-bit	64.79%
	Adaptive 8-Bit (C8F8)	72.12%
	W8B7	76.00%
	W8B6	76.00%
	W8B5	75.00%
	W8B4	72.00%
	W8B3	65.00%
	W8B2	67.00%
	W8B1	68.00%
	W7B8	30.00%
	W6B8	13.00%
	W5B8	28.00%

accuracy when using mixed-precision between weights and biases compared with the pre-trained 32-bit floating point representation, the linear 8-bit integer quantisation and the 8-bit EA-based adaptive integer quantisation methodology from Chapter 5.

Then, the EA-based adaptive integer quantisation methodology is tested on LeNet with MNIST dataset. Based on the previous investigation, the quantisation setting of 8-bit integer representation in convolutional layers with 4-bit integer representation for fully-connected layers is applied to quantise the LeNet on MNIST dataset. In this experiment, the EA-based adaptive integer quantisation is set to a population size of 20 and run for 200 generations. After running the EA-based adaptive integer quantisation for 200 generations, the model achieves a classification accuracy of 99.37%. The result of EA-based adaptive integer quantisation is compared to current approaches.

Table 6.4 Comparison between the EA-based adaptive integer quantisation method and recent approaches in quantisation of CNN on LeNet with MNIST dataset.

Method	Bit width	Data type	Classification error
S. Gupta et al. [104]	14	Fixed point	0.83
	12	Fixed point	0.90%
Deep Compression [14]	8-bit Conv/5-bit FC	Floating point	0.74
TWN [112]	2	Binary	0.65%
SWS [113]	3	Fixed points	0.97%
Bayesian Compression [114]	7-18	Fixed point	1.00%
Proposed	**8-bit Conv/4-bit FC**	**Integer**	**0.63%**

The Table 6.4 illustrates the comparison between the EA-based adaptive integer quantisation and other approaches for quantising LeNet on MNIST dataset. It can be seen from the table, the EA-based adaptive integer quantisation achieves the lowest classification error in quantising LeNet on MNIST dataset, compared to peer competitors. And also, the EA-based adaptive integer quantisation allows the quantised model to process weights and biases in integer representation, compared with most of the quantisaiton methodologies which are applying fixed point for weights and biases.

6.3.3 Combining with Computational Cost Optimisation

The EA-based adaptive integer quantisation demonstrates that the proposed methodology is capable of quantising pre-trained CNNs from 32-bit floating point representation to small bit-width integer representation. Here, the optimised CNNs in Chapter 4 are quantised with 8-bit integer in convolutional layers and 4-bit integer in fully-connected layers.

The Fig. 6.8 illustrates the models' classification accuracy achieved by the proposed methodology. It can be seen from the figure that, although the EA-based adaptive integer quantisation methodology achieves high classification accuracy on standard LeNet architecture after quantising with 8-bit integer in convolutional layers and 4-bit

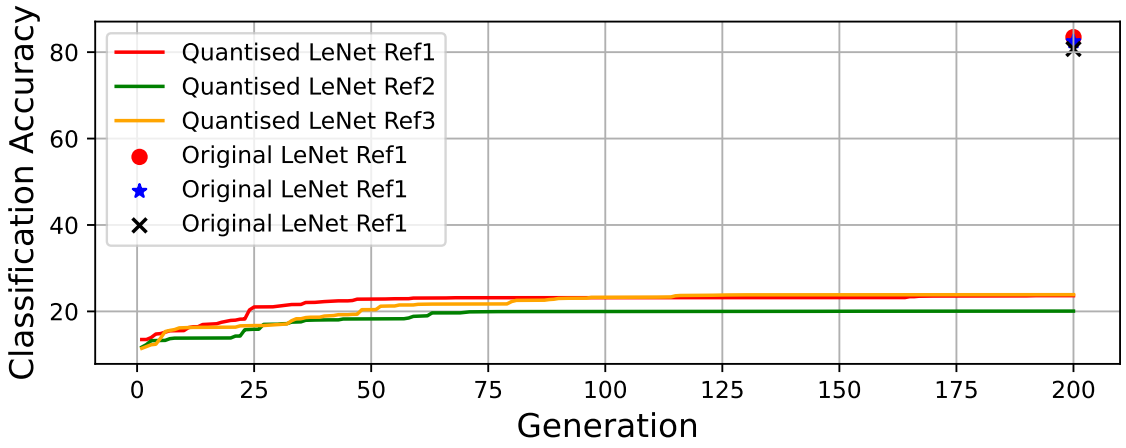


Figure 6.8 Quantising the three reference points of LeNet which are optimised in Chapter 4 on CIFAR-10 dataset with 8-bit representation in convolutional layers and 4-bit integer representation in fully-connected layers.

integer in fully-connected layers, the reference architectures of optimised LeNet in Chapter 4 have very poor classification accuracy after quantisation.

The experimental result suggests that combining the computational cost optimisation with EA-based adaptive integer quantisation methodology cannot make the model achieve a sufficient performance, despite the two methodologies have been demonstrated to significantly improve CNNs' computational cost and model size individually.

6.4 Summary

This chapter has shown how the EA-based adaptive integer quantisation methodology performs using mixed-precision between convolutional layers and fully-connected layers, or between weights and biases for quantising pre-trained CNNs. The experimental results indicate that providing mixed-precision integer representations for quantising pre-trained CNNs can further improve the models' classification accuracy over using the same 8-bit integer setting for all parameters. This suggests that convolutional layers should be quantised with larger integer representation than fully-connected layers.

Typically, applying 8-bit integer representation in convolutional layers and 4-bit integer representation in fully-connected layers achieves the highest classification accuracy (C8F4), compared with other settings, i.e. C8F7, C8F6, C8F5. C8F3, C8F2 and C8F1, of mixed-precision between convolutional layers and fully-connected layers. For the data precision between weights and biases, the experimental results demonstrate that large bit-width should be used for quantising weights and small bit-width integer may be used for quantisation of biases.

The main challenge in this chapter is that it is difficult to combine the EA-based adaptive integer quantisation methodology with the optimised architectures found by computational cost optimisation in Chapter 4. The experimental results illustrate that both optimisation methodologies work well when applied individually. However, when combing the two methodologies together, the model's classification accuracy drops dramatically. This opens up a question for further research into how to best combine the two optimisation methodologies in a beneficial way.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This thesis has developed an optimisation framework that combines evolutionary algorithms (EAs) with convolutional neural networks. The objectives of the thesis are divided into two categories: firstly, to reduce the computational resources costs for processing convolutional neural networks (CNNs) while maintaining CNNs' classification accuracy. Secondly, to provide a quantisation method that can reduce the parameter size with low precision integer representation. Multi-objective evolutionary algorithms (MOEAs) have been applied to reduce the size and number of convolution kernels and expand solution space across various shapes of convolution kernels. Then, an EA-based adaptive integer quantisation is proposed to quantise the weights and bias of CNNs from 32-bit floating point representation to low precision integer representation. It has been demonstrated that these approaches provide opportunities to reduce the size of CNNs and improve their performance.

This work starts with reviewing the background of CNN designs and MOEAs in Chapter 2. The demand identified for modern CNN designs and optimisation is how to reduce model complexity through reduction in the computational resources and memory costs to make the CNNs small and efficient. As a result, EAs can be used to solve optimisation problems that considering single or multiple optimisation objectives. Recent research shows the classification accuracy of CNNs can be improved by optimising the architecture through EA. However, there are limited works focused on reducing the computational resources and memory costs for processing CNNs by evolutionary-inspired techniques.

Chapter 3 demonstrates a MOEA-based framework that is used to trade-off between the computational resource requirement and the model classification accuracy by combining multiple size and shapes of unconventional convolution kernels. The experimental results illustrate that MOEA is able to generate combinations of unconventional kernels to replace the set of one-size square kernels produced by conventional designs and leads

significant reduction in computational resource costs with negligible sacrifice of (and sometimes slightly increased) classification accuracy.

The methodology provides in Chapter 4 is an extension work of the Chapter 3 which is focused on further saving on computational resources costs by reducing the number of convolution kernels in each convolutional layers. In this way, a three-objective MOEA-based optimisation method is used to trade-off between the number of multiplications required for processing a CNN, the total number of convolution kernels a in CNN and the model's classification accuracy. The methodology is evaluated on multiple CNN architectures with various image datasets. As the experimental results show. the optimised CNNs demonstrate significant improvement on saving computational resources and outperform the classification accuracy over standard CNNs.

The results and methods that are developed in Chapter 3 and Chapter 4 have successfully completed **objective 1**, "Develop an optimisation methodology to reduce computational cost of processing feed-forward CNNs with combinations of various unconventional convolution kernel shapes and sizes."

In Chapter 5, an EA-based adaptive integer quantisation methodology is proposed to quantise the weights and biases of pre-trained CNNs from original 32-bit floating representation to 8-bit integer representation. The EA-based adaptive integer quantisation method has been shown to successfully quantise the weights and biases of CNNs. It can be seen from the experimental results that, although the quantised model has a slightly lower classification accuracy than using 32-bit floating point representation, the model size is much smaller than that of the original model.

Chapter 6 carries on the investigation of quantising pre-trained CNNs using mixed-precision integer representations. The investigation is divided into two parts, firstly, different precision of integer presentation between convolutional layers and fully-connected layers is tested. The experimental results show that applying 8-bit integer

for weights and biases in the convolutional layers and 4-bit integer for weights and biases in the fully-connected layers leads to the highest classification accuracy, compared with other combinations. The data representation of weights and biases in fully-connect layers can go down to 2-bit without significant loss in model's classification accuracy. The second part applies different precision for weights and biases. The experimental results suggest that low-precision biases have only a slight impact on the model's classification accuracy. The EA-based adaptive integer quantisation has also been tested on the optimised CNNs from Chapter 4. However, the experimental results indicate that combining both optimisation methodologies leads to poor results in model's classification accuracy.

The methodology and experimental results in Chapter 5 and Chapter 6 corresponds to **objective 2** which is "Devise a methodology that allows adaptive quantisation of both weights and bias from 32-bit floating point representation to lower bit-width integer representation while retaining model classification accuracy". The contributions support that the **objective 2** has been successfully completed.

Review of Hypothesis

At the beginning, the thesis states the main hypothesis:

"Applying evolutionary algorithms to optimise structure of trained CNN models can achieve significant improvements in resource consumption and memory usage while maintaining the classification accuracy of the original models via reducing the number of operations required for processing convolutional layers, and applying low-precision integer data representation for trained CNN models' parameters, i.e. weights and biases."

With following sub-hypotheses:

Sub-hypothesis 1: Unconventional shapes of convolution kernels, e.g. 1×3 and 3×1 kernels, can be used to replace some of the commonly-used square convolution kernels to reduce the computational cost of convolutional layers.

Sub-hypothesis 2: Multi-objective evolutionary algorithms can be used to optimise the computational resource consumption by reducing the size, shape and number of kernels in convolutional layers for specific tasks.

Sub-hypothesis 3: Evolutionary algorithms are capable of quantising CNN weights and biases from their original 32-bit floating point representation to small bit-width integer representation while minimising the loss in classification accuracy.

Sub-hypothesis 4: Applying different bit-widths of integer representation for convolutional layers and fully-connected layers can achieve further reduction in parameter size while minimising the loss in classification accuracy.

The two-objective methodology which is proposed in Chapter 3 combines the MOEA with unconventional convolution kernels. The proposed methodology selects combinations of various unconventional shapes and sizes of 2-D convolution kernels to replace the square kernels in original designs. The proposed methodology generate a trade-off solution between number of multiplication and model's classification accuracy. As the experimental results shows, the number of multiplication required for processing CNNs is significantly reduced with negligible sacrifice of (and sometimes slightly increased) model's classification accuracy. The evidence provided in Chapter 3 can completely support **Sub-hypothesis 1**.

Chapter 4 introduces a three-objective optimisation method that provides trade-offs between number of multiplications, number of kernels in convolutional layers and model's classification accuracy. The proposed methodology combining different number and multiple size of low-computation unconventional convolution kernels.

The methodology has been tested on various CNNs with multiple datasets. The experimental results in Chapter 4 provide reasonable support for **Sub-hypothesis 2**.

The EA-based adaptive integer quantisation methodology proposed in Chapter 5 quantises the weights and biases in pre-trained from 32-bit floating point representation to 8-bit integer representation. Then, Chapter 6 applies mixed-precision integer representation between convolutional layers and fully-connected layers of pre-trained CNNs to further reduce the parameter size. An investigation of mix-precision integer representation between weights and biases is also carried out in Chapter 6. The resultant evidence in Chapter 5 and Chapter 6 fully support the **Sub-hypothesis 3** and **Sub-hypothesis 4**.

Critical Discussions

This PhD thesis has successfully pushed the research boundary showing that evolutionary algorithms are promising techniques to optimise CNN designs in the context of reducing computational resources cost and parameter size. However, there are some limitations in this PhD work which may be improved in the future.

- As a population-based optimisation methodology, the proposed evolutionary optimisation techniques requires a long time to run.
- The proposed computational cost optimisation and data representation optimisation demonstrate significant improvement in reducing computational cost and parameter size, respectively. However, in Chapter 6, the combination of two optimisation methodologies shows poor results in models' classification accuracy. An initial guess as to what is causing this issue is that the optimised models have fewer kernels than the original models, and the kernel size is also smaller in the optimised model. In order to extract enough features from inputs, each

kernel might need a large range to represent data, which might result in the representation range of an integer not being enough to cover the range.

7.2 Future Work

Based on the findings in this thesis, there are few research avenues which can be explored in the future.

Runtime Optimisation for Evolutionary Algorithms

The evolutionary algorithms are population-based methods, which require numerous evaluations and computing resources essential for implementing evolutionary algorithms. Typically, when applying evolutionary techniques to optimise CNN designs, each model needs to be trained and validated individually. For the proposed computational cost optimisation methodology, the most time-consuming and computation-costly part is training CNNs that large amount of data in training dataset have to be processed for certain epochs. Therefore, reducing the runtime when training CNNs will accelerate the proposed optimisation methodology. Possible research direction can be, for example, finding out how many epochs are enough for training target CNNs. When CNNs are trained for enough epochs that the performance might not be improved by further training, the training process can be terminated early to save time and computational costs, instead of training the model for a fixed epochs.

The second way for accelerating the optimisation process would be *parallelisation*. All experiments in Chapter 3 and Chapter 4 are implemented on a single GPU. In this case, only one individual can be trained at a time. Therefore, if there are more GPU resources available, all individual in each generation can be trained together on separate GPUs. For example, in Chapter 3 and Chapter 4, the population contains

25 individuals. The optimisation loop could apply 25 GPUs to train these individuals together. In this case, the runtime for the proposed method can be accelerated up to 25×, if the memory speed and bandwidth are also enough.

Combining Computational Cost Optimisation and Data Representation Optimisation

One of the experiments in Chapter 6 investigates how the models' classification accuracy will be affected by combining the computational cost optimisation methodology and adaptive integer quantisation methodology. However, the results demonstrate that there is a large gap between the original model (optimised by computational cost optimisation in Chapter 4) and the quantised model. The potential cause has been discussed previous section which could be an interesting research avenue for combining two optimisation methodologies.

Implementing the Optimised CNNs into Hardware

The experimental results in this PhD work demonstrate significant saving in content of number of multiplication in Chapter 3 and Chapter 4. The reduction in number of multiplications can be regarded as multiply-accumulate operations in hardware implementation. Therefore, implementing CNNs which optimised by this PhD work into hardware platform to test the performance will be one of the future work. Moreover, an efficient approximate multiply-accumulate array [1] has been proposed to replace the conventional multiply-accumulator during this PhD work. How to implement the optimised CNNs into the approximate multiply-accumulate array [1] and optimise dataflow for processing the typical optimised CNNs can be one of future research direction.

Evolutionary Optimisation for Recurrent Neural Networks

The optimisation methodologies proposed in this PhD work is focusing on reducing the computational cost and models' size of CNNs. Recurrent neural networks are another type of artificial neural works which also require millions of multiplications and gigabytes of memory to process data. Since the evolutionary technique demonstrated significant improvement in optimising CNNs, how to apply the evolutionary algorithms to optimise the computational cost and parameter size of recurrent neural networks can be an interesting investigation for the future.

Abbreviations

(1+1)-ES (1+1) Evolutionary Strategy

($\mu + \lambda$)-ES ($\mu + \lambda$) Evolutionary Strategy

(μ, λ)-ES (μ, λ) Evolutionary Strategy

1-D One-dimension

2-D Two-dimensions

3-D Three-dimensions

ACBs Asymmetric Convolution Blocks

AI Artificial Intelligence

ANN Artificial Neural Network

BNN Binarized Neural Network

BN Batch Normalisation

CMA-ES Covarience Matrix Adaptation Evolution strategy

CNN Convolutional Neural Network

DL Deep Learning

DNN Deep Neural Network

EA Evolutionary Algorithm

FCNN Fully-Connected Neural Network

FFT Fast Fourier Transformation

FPGA Field-Programmable Gate Array

GA Genetic Algorithm

LRN Local Response Normalisation

MAC Multiply-accumulation

MOEA Multi-objective Evolutionary Algorithm

MOGA Multi-objective Genetic Algorithm

NSGA-II Non-dominated Sorting Genetic Algorithm-II

NSGA Non-dominated Sorting Genetic Algorithm

SGD Stochastic Gradient Descent

VEGA Vector Evaluated Genetic Algorithm

VQ Vector Quantisation

References

[1] Z. Wang, M. A. Trefzer, S. J. Bale, and A. M. Tyrrell, "Approximate multiply-accumulate array for convolutional neural networks on fpga," in *2019 14th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*. IEEE, 2019, pp. 35–42.

[2] Z. Wang, M. A. Trefzer, S. Bale, and A. M. Tyrrell, "Adaptive integer quantisation for convolutional neural networks through evolutionary algorithms," in *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2021, pp. 1–7.

[3] Z. Wang, M. A. Trefzer, S. J. Bale, and A. M. Tyrrell, "A multi-objective evolutionary approach for efficient kernel size and shape for cnn," in *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2022, pp. 1–8.

[4] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[6] 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.

[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 *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[8] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[9] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[11] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning*. PMLR, 2019, pp. 6105–6114.

[12] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

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

[14] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

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

[16] M. Bhuvaneswari, *Application of evolutionary algorithms for multi-objective optimization in VLSI and embedded systems*. Springer, 2014.

[17] C. A. C. Coello, G. B. Lamont, D. A. Van Veldhuizen *et al.*, *Evolutionary algorithms for solving multi-objective problems*. Springer, 2007, vol. 5.

[18] 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.

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

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

[21] 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.

[22] Y. Bengio *et al.*, "Learning deep architectures for ai," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

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

[24] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, no. 1, 2013, p. 3.

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

[26] S. Belharbi, "Neural networks regularization through representation learning," *arXiv preprint arXiv:1807.05292*, 2018.

[27] 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.

[28] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[29] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[30] 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.

[31] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[32] N. Bjorck, C. P. Gomes, B. Selman, and K. Q. Weinberger, "Understanding batch normalization," *Advances in neural information processing systems*, vol. 31, 2018.

[33] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 2014, pp. 675–678.

[34] F. Tu, S. Yin, P. Ouyang, S. Tang, L. Liu, and S. Wei, "Deep convolutional neural network architecture with reconfigurable computation patterns," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 8, pp. 2220–2233, 2017.

[35] M. A. Trefzer and A. M. Tyrrell, "Evolvable hardware," *From Practice to Application. Springer*, 2015.

[36] A. E. Eiben, J. E. Smith *et al.*, *Introduction to evolutionary computing*. Springer, 2003, vol. 53.

[37] J. Brownlee, *Clever algorithms: nature-inspired programming recipes*. Jason Brownlee, 2011.

[38] F. Hoffmeister and T. Bäck, "Genetic algorithms and evolution strategies: Similarities and differences," in *International Conference on Parallel Problem Solving from Nature*. Springer, 1990, pp. 455–469.

[39] A. P. Engelbrecht, *Computational intelligence: an introduction*. John Wiley & Sons, 2007.

[40] R. Hinterding, "Gaussian mutation and self-adaption for numeric genetic algorithms," in *Proceedings of 1995 IEEE International Conference on Evolutionary Computation*, vol. 1. IEEE, 1995, p. 384.

[41] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies–a comprehensive introduction," *Natural computing*, vol. 1, no. 1, pp. 3–52, 2002.

[42] N. Hansen, "The cma evolution strategy: A tutorial," *arXiv preprint arXiv:1604.00772*, 2016.

[43] V. Pareto, *Cours d'économie politique*. Librairie Droz, 1964, vol. 1.

[44] 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.

[45] T. Back, M. Emmerich, and O. Shir, "Evolutionary algorithms for real world applications [application notes]," *IEEE Computational Intelligence Magazine*, vol. 3, no. 1, 2008.

[46] J. D. Schaffer, "Multiple objective optimization with vector evaluated genetic algorithms," in *Proceedings of the First International Conference on Genetic Algorithms and Their Applications, 1985*. Lawrence Erlbaum Associates. Inc., Publishers, 1985.

[47] C. M. Fonseca, P. J. Fleming *et al.*, "Genetic algorithms for multiobjective optimization: Formulationdiscussion and generalization." in *Icga*, vol. 93, no. July, 1993, pp. 416–423.

[48] N. Srinivas and K. Deb, "Muiltiobjective optimization using nondominated sorting in genetic algorithms," *Evolutionary computation*, vol. 2, no. 3, pp. 221–248, 1994.

[49] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: from architectures to learning," *Evolutionary intelligence*, vol. 1, no. 1, pp. 47–62, 2008.

[50] S. Fujino, N. Mori, and K. Matsumoto, "Deep convolutional networks for human sketches by means of the evolutionary deep learning," in *2017 Joint 17th World Congress of International Fuzzy Systems Association and 9th International Conference on Soft Computing and Intelligent Systems (IFSA-SCIS)*. IEEE, 2017, pp. 1–5.

[51] A. Singh, S. Saha, R. Sarkhel, M. Kundu, M. Nasipuri, and N. Das, "A genetic algorithm based kernel-size selection approach for a multi-column convolutional neural network," *arXiv preprint arXiv:1912.12405*, 2019.

[52] A. Dahou, M. A. Elaziz, J. Zhou, and S. Xiong, "Arabic sentiment classification using convolutional neural network and differential evolution algorithm," *Computational intelligence and neuroscience*, vol. 2019, 2019.

[53] B. Cheung and C. Sable, "Hybrid evolution of convolutional networks," in *2011 10th International Conference on Machine Learning and Applications and Workshops*, vol. 1. IEEE, 2011, pp. 293–297.

[54] Y. Bi, B. Xue, and M. Zhang, "An evolutionary deep learning approach using genetic programming with convolution operators for image classification," in *2019 IEEE congress on evolutionary computation (CEC)*. IEEE, 2019, pp. 3197–3204.

[55] S. Gibb, H. M. La, and S. Louis, "A genetic algorithm for convolutional network structure optimization for concrete crack detection," in *2018 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2018, pp. 1–8.

[56] L. M. Zhang, "A new compensatory genetic algorithm-based method for effective compressed multi-function convolutional neural network model selection with multi-objective optimization," *arXiv preprint arXiv:1906.11912*, 2019.

[57] J. Prellberg and O. Kramer, "Lamarckian evolution of convolutional neural networks," in *International Conference on Parallel Problem Solving from Nature*. Springer, 2018, pp. 424–435.

[58] M. G. B. Calisto and S. K. Lai-Yuen, "Self-adaptive 2d-3d ensemble of fully convolutional networks for medical image segmentation," in *Medical Imaging 2020: Image Processing*, vol. 11313. SPIE, 2020, pp. 459–469.

[59] M. Baldeon-Calisto and S. K. Lai-Yuen, "Adaresu-net: Multiobjective adaptive convolutional neural network for medical image segmentation," *Neurocomputing*, vol. 392, pp. 325–340, 2020.

[60] B. Wang, Y. Sun, B. Xue, and M. Zhang, "Evolving deep neural networks by multi-objective particle swarm optimization for image classification," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019, pp. 490–498.

[61] B. Fielding and L. Zhang, "Evolving image classification architectures with enhanced particle swarm optimisation," *IEEE Access*, vol. 6, pp. 68 560–68 575, 2018.

[62] M. Loni, A. Majd, A. Loni, M. Daneshtalab, M. Sjödin, and E. Troubitsyna, "Designing compact convolutional neural network for embedded stereo vision systems," in *2018 IEEE 12th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*. IEEE, 2018, pp. 244–251.

[63] D. Song, C. Xu, X. Jia, Y. Chen, C. Xu, and Y. Wang, "Efficient residual dense block search for image super-resolution," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 07, 2020, pp. 12 007–12 014.

[64] M. Suganuma, M. Kobayashi, S. Shirakawa, and T. Nagao, "Evolution of deep convolutional neural networks using cartesian genetic programming," *Evolutionary computation*, vol. 28, no. 1, pp. 141–163, 2020.

[65] T. Hassanzadeh, D. Essam, and R. Sarker, "Evou-net: an evolutionary deep fully convolutional neural network for medical image segmentation," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, 2020, pp. 181–189.

[66] E. Miahi, S. A. Mirroshandel, and A. Nasr, "Genetic neural architecture search for automatic assessment of human sperm images," *Expert Systems with Applications*, vol. 188, p. 115937, 2022.

[67] N. Mitschke, M. Heizmann, K.-H. Noffz, and R. Wittmann, "Gradient based evolution to optimize the structure of convolutional neural networks," in *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE, 2018, pp. 3438–3442.

[68] Z. Yang, Y. Wang, X. Chen, B. Shi, C. Xu, C. Xu, Q. Tian, and C. Xu, "Cars: Continuous evolution for efficient neural architecture search," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1829–1838.

[69] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun, "Single path one-shot neural architecture search with uniform sampling," in *European conference on computer vision*. Springer, 2020, pp. 544–560.

[70] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan, "A survey on evolutionary neural architecture search," *IEEE transactions on neural networks and learning systems*, 2021.

[71] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[72] T. Elsken, J.-H. Metzen, and F. Hutter, "Simple and efficient architecture search for convolutional neural networks," *arXiv preprint arXiv:1711.04528*, 2017.

[73] 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.

[74] 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.

[75] 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.

[76] 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.

[77] 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.

[78] 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.

[79] 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.

[80] 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.

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

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

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

[84] 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.

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

[86] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

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

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

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

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

[91] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu, "Deeply-supervised nets," in *Artificial intelligence and statistics*. PMLR, 2015, pp. 562–570.

[92] A. Y. Ng, "Feature selection, l 1 vs. l 2 regularization, and rotational invariance," in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 78.

[93] D. Kang and C. W. Ahn, "Efficient neural network space with genetic search," in *International Conference on Bio-Inspired Computing: Theories and Applications*. Springer, 2019, pp. 638–646.

[94] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Evolving deep convolutional neural networks for image classification," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 2, pp. 394–407, 2019.

[95] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *arXiv preprint arXiv:1806.08342*, 2018.

[96] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[97] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European conference on computer vision*. Springer, 2016, pp. 525–542.

[98] A. Gersho and R. M. Gray, *Vector quantization and signal compression*. Springer Science & Business Media, 2012, vol. 159.

[99] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," *arXiv preprint arXiv:1412.6115*, 2014.

[100] J. Su, "Artificial neural networks acceleration on field-programmable gate arrays considering model redundancy," 2018.

[101] M. Denil, B. Shakibi, L. Dinh, N. De Freitas *et al.*, "Predicting parameters in deep learning," in *Advances in neural information processing systems*, 2013, pp. 2148–2156.

[102] K. O. Stanley, "Efficient evolution of neural networks through complexification," Ph.D. dissertation, 2004.

[103] M. A. Trefzer, T. Kuyucu, J. F. Miller, and A. M. Tyrrell, "Image compression of natural images using artificial gene regulatory networks," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. ACM, Jul. 2010, pp. 595–602.

[104] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International conference on machine learning*. PMLR, 2015, pp. 1737–1746.

[105] D. Lin, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *International conference on machine learning*. PMLR, 2016, pp. 2849–2858.

[106] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4820–4828.

[107] M. Sit, R. Kazami, and H. Amano, "Fpga-based accelerator for losslessly quantized convolutional neural networks," in *2017 International Conference on Field Programmable Technology (ICFPT)*. IEEE, 2017, pp. 295–298.

[108] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[109] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 65–74.

[110] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, "Fp-bnn: Binarized neural network on fpga," *Neurocomputing*, vol. 275, pp. 1072–1086, 2018.

[111] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Icml*, 2010.

[112] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint arXiv:1605.04711*, 2016.

[113] K. Ullrich, E. Meeds, and M. Welling, "Soft weight-sharing for neural network compression," *arXiv preprint arXiv:1702.04008*, 2017.

[114] C. Louizos, K. Ullrich, and M. Welling, "Bayesian compression for deep learning," *Advances in neural information processing systems*, vol. 30, 2017.