DEEP NEURAL NETWORK GENERATION FOR IMAGE CLASSIFICATION WITHIN RESOURCE-CONSTRAINED ENVIRONMENTS USING EVOLUTIONARY AND HAND-CRAFTED PROCESSES

T Lawrence

PhD

2021

DEEP NEURAL NETWORK GENERATION FOR IMAGE CLASSIFICATION WITHIN RESOURCE-CONSTRAINED ENVIRONMENTS USING EVOLUTIONARY AND HAND-CRAFTED PROCESSES

TOM LAWRENCE

A thesis submitted in partial fulfilment of the requirements of the University of Northumbria at Newcastle for the degree of Doctor of Philosophy

Research undertaken in the Faculty of Engineering and Environment and in collaboration with Ocucon, Newcastle upon Tyne

December 2021

Abstract

Constructing Convolutional Neural Networks (CNN) models is a manual process requiring expert knowledge and trial and error. Background research highlights the following knowledge gaps. 1) existing efficiency-focused CNN models make design choices that impact model performance. Better ways are needed to construct accurate models for resource-constrained environments that lack graphics processing units (GPU's) to speed up model inference time such as CCTV cameras and IoT devices. 2) Existing methods for automatically designing CNN architectures do not explore the search space effectively for the best solution and 3) existing methods for automatically designing CNN architectures do not exploit modern model architecture design patterns such as residual connections. The lack of residual connections means the model depth is limited owing to the vanishing gradient problem. Furthermore, existing methods for automatically designing CNN architectures adopt search strategies that make them vulnerable to local minima traps.

Better techniques to construct efficient CNN models, and automated approaches that can produce accurate deep model constructions advance many areas such as hazard detection, medical diagnosis and robotics in both academia and industry.

The work undertaken during this research are 1) the proposal of an efficient and accurate CNN architecture for resource-constrained environments owing to a novel block structure containing 1x3 and 3x1 convolutions to save computational cost, 2) proposed a particle swarm optimization (PSO) method of automatically constructing efficient deep CNN architectures with greater accuracy by proposing a novel encoding and search strategy, 3) proposed a PSO based method of automatically constructing deeper CNN models with improved accuracy by proposing a novel encoding scheme that employs residual connections, in novel search mechanism that follows the global and neighbouring best leaders.

The main findings of this research are 1) the proposed efficiency-focused CNN model outperformed MobileNetV2 by 13.43% in respect to accuracy, and 39.63% in respect to efficiency, measured in floating-point operations. A reduction in floating-point operations means the model has the potential for faster inference times which is beneficial to applications within resource-constrained environments without GPU's such as CCTV cameras. 2) the proposed automatic CNN generation technique outperformed existing methods by 7.58% in respect to accuracy and a 63% improvement in search time efficiency owing to the proposal of more efficient architectures speeding up the search process and 3) the proposed automatic deep residual CNN generation method improved model accuracy by 4.43% when compared against related studies owing to deeper model construction and improvements in the search process. The proposed search process embeds human knowledge of constructing deep residual networks and provides constraint settings which can be used to limit the proposed models depth and width. The ability to constrain a models depth and width is important as it ensures the upper bounds of a proposed model will fit within the constraints of resource-constrained environments.

List of Publications

Published peer-reviewed papers:

- **T. Lawrence**, L. Zhang, K. Rogage and C. Lim, "Evolving Deep Architecture Generation with Residual Connections for Image Classification Using Particle Swarm Optimization," *Sensors*, vol. 21, Article 7936, 2021.
- **T. Lawrence**, L. Zhang, C. Lim and E Phillips, "Particle Swarm Optimization for Automatically Evolving Convolutional Neural Networks for Image Classification," *IEEE Access*, vol. 9, pp. 14369-14386, 2021
- **T. Lawrence**, L. Zhang, "IoTNet: An Efficient and Accurate Convolutional Neural Network for IoT Devices," *Sensors*, vol. 19, Article 5541, 2019.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to offer my special thanks to Ocucon Ltd, specifically Simon Gardner for the opportunity to work on such an interesting project. I have gained valuable experience and have thoroughly enjoyed withing with you and the Ocucon team.

I would like to thank my supervisor Li Zhang. Li's encouragement, support and guidance have greatly improved my research and academic writing skills, resulting in three journal publications. I am thankful for the support provided to me which went above and beyond any expectations. Li is exceptionally dedicated to academia and is one of the hardest working individuals I have had the pleasure to work with.

I would like to thank Kay Rogage who took on the position as my principal supervisor in the final months of my PhD project. Kay has provided an invaluable fresh eyes perspective to my project, providing valuable guidance, support and feedback.

I would like to extend my sincere thanks to my second supervisor Wai Lok Woo. Lok has provided great feedback and support to further shape my academic writing.

I am grateful to the Intensive Industrial Innovation Programme and Northumbria University, whose financial support has made it possible for me to conduct my research. Without such support, it would not have been possible for me to undertake such a project.

Declaration

I declare that the work contained in this thesis has not been submitted for any other award and that it is all my own work. I also confirm that this work fully acknowledges opinions, ideas and contributions from the work of others. The work was done in collaboration with Ocucon, Newcastle upon Tyne.

Any ethical clearance for the research presented in this commentary has been approved. Approval has been sought and granted through the Researcher's submission to Northumbria University's Ethics Online System on 22 December 2021.

I declare that the Word Count of this Thesis is 32895 words

Name: Tom Lawrence

Date: 3 January 2022

1 Introduction

Convolutional Neural Networks (CNNs) have proved revolutionary in computer vision applications as they consistently outperform traditional models or even humans at image recognition tasks. Designing CNNs from scratch remains a challenging task that requires expert domain knowledge and significant levels of trial and error. An active research area exists which has proposed many interesting architectures, with the main aim of maximizing accuracy on competition data sets. The wider community benefits from such knowledge by reapplying existing models to new problem domains. The new problem domains are commonly less complex and more focused problems, so the resulting models are often overly complex for the task at hand. In extremely resource constrained environments, a similar approach is adopted by applying a previously proposed efficiency focused CNN model. Efficiency focused models contain strategies that increase computational efficiency at great expense to accuracy. When trying to tackle a new problem, a custom model specifically designed for the task at hand would result in the most optimal model. Attempts have been made to propose methods that automate the CNN design process. Work in the field of automated CNN generation techniques is still a new research area. The existing state of the art methods focuses on relatively basic search strategies that do not effectively explore the available search space. Moreover, some techniques require expert domain knowledge to implement such a search strategy and do not incorporate modern CNN design techniques. Owing to the aforementioned knowledge gaps, this research has been funded by the European Regional Development Fund, in collaboration with an industry partner Ocucon Ltd to address these gaps. Ocucon Ltd is a software as a service company that specialises in computer vision based technologies. Such computer vision based technologies involve the development of Convolutional Neural Networks (CNNs) models. The CNN models are applied primarily within the CCTV industry. The main customer of Ocucon is the retail industry which is price sensitive and fast-paced. The deployment of CNN models at Ocucon are either conducted as edge deployments on resource-constrained devices such as IoT devices and CCTV camera-related equipment or centralised at a data centre. Hardware requirements scale with model complexity and complex model is more challenging to adopt and scale meaning that there is a real need to address the aforementioned knowledge gaps. This research aims to address the aforementioned academic and industry knowledge gaps by:

- Proposing an efficiency focused CNN model that reduces computational complexity, whilst maximizing accuracy so that more accurate models can be deployed in constrained environments such as CCTV cameras.
- Proposing a PSO based search strategy for automatically designing efficient networks for the data set at hand, that produces more accurate models when compared against the current state-of-the-art approaches.
- Proposing a PSO based search strategy capable of generating deeper CNN architec-

tures owing to the inclusion of residual connections.

1.1 Introduction to CNN Models

Traditionally, state-of-the-art CNNs such as LeNet [1], AlexNet [2] and ResNet [3], Wide ResNet [4] and DenseNet [15] are all designed by manual processes to compete on benchmark data sets such as ImageNet [16] which is a large data set containing approximately 1 million 469x387 colour images and 1,000 object classes, MS COCO [17] which contains 328,000 640×480 colour images with 80 object classes, CIFAR-10 which consists of 60000 32x32 colour images containing 10 different classes and CIFAR-100 [18] which consists of 60000 32x32 colour images containing 100 different classes. The resulting models are often repurposed to solve challenges in new problem domains using methods such as transfer learning [19]. For example, [20] applied transfer learning to train a DenseNet network to perform keyword spotting on smart terminal devices, while the ResNet architecture was adopted by [21] for medical diagnosis. Studies such as [22] adopted a CNN model to perform gas identification as part of the wider research area of electronic noses (ENs).

Transfer learning involves taking a pre-trained model and training some of the model layers further using a new data set [23]. Such a transfer learning process relies heavily on existing certain well-known deep architectures. They are often overly complex as the original model intention has been focused on obtaining state-of-the-art performance on complex and varied large-scale data sets. On the other hand, hand-crafting a new model from scratch requires specialist knowledge and trial-and-error owing to a vast number of design choices and hyperparameter settings. It is a bottleneck of designing a network for a new application domain.

A typical CNN can be divided into several specific types of layers. In general, each layer has a specific type. There are three distinctive types of layers, i.e. 1) convolutional layers for extracting features such as edges, 2) average or max pooling layers for reducing the feature map sizes, and 3) fully connected layers for model classification. The convolutional layers perform convolutional operations on the input images for deep feature learning while the pooling layers down-sample an input dimension to achieve better spatial invariance by reducing the feature map size and capturing invariances in image-like data [24]. A reduction in the size of a feature map also results in fewer parameters, mitigating the overfitting issue [25] and lowering the computational cost. The final layer is a fully connected layer for classification purposes.

A convolution is performed by sliding a kernel of size K over an input feature map of size M and taking the element-wise product. The distance at which the kernel is moved is known as stride S and the input feature map can also be given a P padding. The process can be repeated multiple times using different kernels to increase the number of output channels, which is denoted as the model width, e.g. in Wide Resnet [4]. Using larger strides reduces the output dimensions, however a common practice is to perform all convolution operations based on the same convolution formulation, i.e., use appropriate padding to ensure matching between the input and output dimensions. The required padding for each side of the input volume to perform the same convolution formulation can be found using Equation 1, with the most common setting of $S = 1$.

$$P = \frac{K - S}{2} \tag{1}$$

1.2 Background

1.2.1 Efficiency Focused CNN Architecture Background and Gap

Many studies within computer vision focus on improving accuracy by designing a new state-of-the-art model, typically only ever constrained by the resources available on high-end graphics cards. State-of-the-art models are typically very deep. This is because the network generalization capability is enhanced, as the network goes deeper. However, the downside to deep models is that even the most cutting-edge and efficient state-of-the-art models, such as EfficientNet [26], still contain millions of parameters and billions of FLOPs. Models such as these require significant computational resources to execute, and exhibit diminishing returns when scaling. For example, Related works such as WideResnet [4] has shown a model containing approximately 1 million parameters spread over 40 layers takes 61.3 seconds to compute per training epoch on CIFAR-10, but when organised over 22 wider model layers, the time taken is reduced to 54.45 seconds per training epoch. The model with 40 layers produced a 0.28% accuracy improvement over the 22 model configuration on the CIFAR-10 data set. This indicates that a large increase in model depth produces a small improvement in accuracy. Therefore, such an approach results in a very large yet extremely accurate model. When deploying a CNN model within a resource-constrained environment such as IoT devices or smartphones, it becomes critical to find a balance between model accuracy and computational cost to ensure the model will function well within resource limited environments. When finding such a trade-off, two main approaches exist. The first approach is to scale down a large model to fit the constraints of the target device as it seems reasonable to assume that if a large increase in the size of a state-of-the-art model results in a small improvement in accuracy, then a large reduction in model size would result in a small loss of performance. While this is true to an extent, the point at which accuracy starts to drop rapidly occurs while the model is still very large. This is because the state-of-the-art accuracy-focused models contain strategies that help such networks overcome the types of issues encountered during training, such as overfitting. To scale such a model down sufficiently enough for tightly constrained environments, expert knowledge and trial-and-error would be required. The second approach is to design models specifically for computational constrained environments. As an example, efficiency-focused models include MobileNet [5], ShuffleNet [6] and EffNet [9]. Such models excel at delivering far greater accuracy than would be possible by significantly scaling down a large model. This is achieved by making design choices that reduce the computational cost, often by performing convolutions as depth-wise separable convolutions instead of normal convolutions employed by their larger model counterparts. A distinction between the above two types of convolutions (i.e., normal and depth-wise separable convolutions) is discussed comprehensively in Section 3.2.

The weakness of the above two approaches identifies the following gap in knowledge:

- Scaling down a large state-of-the-art CNN model is not a sufficient approach for resource constrained environments.
- Existing efficiency focused CNN models do not trade model efficiency with accuracy well. They include strategies such as depth-wise separable convolutions which negatively impact feature model accuracy.

1.2.2 Automating CNN Architecture Generation Background and Gap

Designing Convolutional Neural Networks (CNNs) for a new domain is very challenging. The design task requires expert knowledge and significant manual trial-and-error to produce a model which performs well during testing and generalizes well on unseen data. The reason designing a CNN is challenging is because of the important design choices which must be made. Such design choices include selecting a suitable kernel size owing to its importance in adjusting the receptive field [27]. Another important step is balancing the depth of the network by adding or removing layers as well as increasing or decreasing the number of filters per layer within the network [4].

Because of these challenges, a popular approach when tackling a new problem is to use an existing state-of-the-art hand-crafted network as a starting point. The disadvantage of such an approach is that the network is often overly large for the task at hand. This is because the design choices made during the construction of the hand-crafted networks were to maximize accuracy on challenging competition data sets. Such deep state-of-the-art networks also require a substantial amount of training data than those used for training shallower networks [28]. When tackling new and novel image classification problems, it is usually a challenging task to obtain a sufficient amount of data. To combat this, methods such as pruning convolutional kernels have been proposed, to slim down networks by up to 10 times smaller [29], measured as a reduction of the convolutional kernels of the final network architecture. A weakness of the pruning approach is that it requires a suitable network as the starting point. On the other hand, transfer learning is another mechanism used to leverage state-of-the-art CNN architectures in new application domains. Many existing studies employed transfer learning to fine-tune a pre-trained network for undertaking a task in a new domain. As an example, a pre-trained ResNet network was re-trained by [30] for tackling malicious software classification problems.

Owing to the aforementioned weaknesses and challenges, a new research era is opening up which aims to address the research question of how to automatically design deep CNN architectures for tackling problems in new domains. Such research efforts have resulted in some impressive developments in the field where evolutionary algorithms are used to automatically devise CNN models from scratch, psoCNN [10] and GeNET [31], where Particle Swarm Optimization (PSO) [32] and the Generic Algorithms (GA) were used for evolving deep network generation.

The concept of automatically evolving CNNs refers to an automatic procedure for the generation of CNN models with diverse architectures for tackling different problems [33] [13] [12]. Many existing studies on automatic CNN architecture generation adapt the existing meta-heuristic algorithms [34] for deep architecture search [33] [35]. Examples include Genetic Algorithm (GA) [36] [37] which employs selection, crossover and mutation operations to evolve the search process. On the other hand, Particle Swarm Optimization (PSO) [32] utilizes the personal and global best solutions to explore the search space. IPPSO [13] has been one of the first studies to apply PSO for deep network design. Inspired by network IP addresses, the method adopts a new encoding scheme to overcome network representation constraints so that complex models can be easily encoded. EvoCNN [12] presents a GA-based approach that uses a variable-length gene encoding strategy to represent the building blocks of a CNN model. Because of the impressive global search capabilities of PSO and its computational efficiency (e.g. as compared with the GA), psoCNN [10] has been proposed recently for deep network generation. The model adapts a traditional PSO algorithm to behave more like the GA, where a particle position is updated by copying the layers at random from either the personal or global best solutions. Although psoCNN outperforms a number of existing methods, e.g. IPPSO and EvoCNN, for deep architecture generation, the model suffers from a number of constraints including the need to define rules to ensure solutions remain valid. PSO based approaches such as psoCNN also contain a movement strategy that copies blocks from the global or personal best solutions, meaning that intermediate positions are not explored.

The existing body of knowledge shows that PSO based approaches for automatically generating CNN models are the current state-of-the-art, however the following issues highlight gaps in knowledge still exist:

- State-of-the-art PSO based architecture generation techniques such as psoCNN [10] do not perform a detailed search. Instead, particle positions are updated by copying aspects from other global or personal best solutions. Such an approach means that intermediate positions are not explored, degrading the search quality.

- PSO based architecture generation techniques such as psoCNN [10] require complex rules to guard against invalid CNN architectures. Invalid model guarding rules include removing any fully connected layer from the start of the network and limiting how many pooling layers a network can contain to ensure downsampling is not performed excessively. While such rules do ensure valid networks are generated, the mechanism of defining rules that manually move a particle interrupts the natural search process. Manual rules also increase the complexity of implementing such a solution, as all edge cases must be considered.

1.2.3 Automating Residual CNN Architecture Generation Background and Gap

Existing state-of-the-art CNN architecture construction techniques fall into two main categories which are 1) CNN architecture generation and 2) CNN architecture optimization. Existing state-of-the-art CNN architecture generation techniques such as sosCNN [14], psoCNN [10] and IPPSO [13] perform a search capable of optimizing important hyperparameters such as kernel size and pooling types. The weakness of such studies is that they perform architecture searches on relatively basic CNN architectures i.e do not take advantage of advanced construction techniques like Residual connections. Residual connections were proposed in ResNet [3] as a way to address the vanishing gradient problem. Residual connections at a high level organise a network into blocks. Each block contains two convolutional layers. The output of the previous block gets added to the output of the current block. Such a connection means that gradients have a path to flow which passes through less convolutional layers, making them less likely to become 0, or vanish. The authors of ResNet showed that such an approach enabled deeper model construction. Further details of residual connections is provided in Section 2. The lack of residual connections in studies relating to CNN architecture construction makes them susceptible to the vanishing gradient problem [3] [38]. Existing methods for architecture optimization such as [15] [39] [40] do make use of advanced construction techniques like residual connections by optimizing existing state-of-the-art models, however, they limit the search by not optimizing hyperparameters such as the kernel sizes or pooling types. This limits the variety of the generated networks. In addition, most of the existing search methods, such as sosCNN [14] and psoCNN [10], adopt the traditional PSO operations by using the personal and global best solutions to guide the search process. Therefore, their search processes are more likely to be trapped in local optima owing to the dominance of a single global best leader [41]

The following gaps are highlighted:

- Existing CNN generation techniques do not exploit advanced CNN techniques such as residual connections.
- Existing CNN optimization techniques perform a limited search that limits the variety of the generated networks.
- Existing PSO based CNN generation techniques may become trapped in local optima due to following the personal and global best leaders.

1.3 Motivations and Research Questions

Neural networks and CNN models perform classification tasks by passing data through a series of connected layers. Each layer performs a floating point operation before passing its output to the next layer. The larger the network, the longer this process takes as the hardware running the CNN model must perform more calculations to compute the output owing to the greater number of layers. Research such as [3] has shown that larger networks are more accurate, however, larger networks are slower for the aforementioned reasons.

To accelerate the process of performing floating point operations, research such as [42] first proposed using a graphics processing unit (GPU) rather than a central processing unit (CPU) for computing the calculations within a network. The authors showed that using GPUs yields over 3X speedup for both training and testing when compared to a 3GHz P4 CPU. GPUs are commonly found on desktop computers, and GPU equipped servers are available but low powered edge devices such as CCTV cameras, often powered over Ethernet cables with limited power available [43] do not have GPU capabilities because of cost and power limitations, they are therefore heavily resource constrained. Moreover, scaling up the throughput of a CNN model in a production environment requires additional hardware, which increases cost. Cost constraints the commercial feasibility of a CNN based solution. This research is heavily motivated by a need to design more efficient CNN architectures so more accurate models can be deployed to research contained environments such as CCTV cameras and production data centers.

Designing a CNN model from scratch requires expert knowledge and significant trial and error. Rather than designing a new model from scratch to solve problems in a new domain, a common approach is to train some of the final layers of an existing state-of-the-art model on a new data set in a process called transfer learning [23] which has two main weaknesses. 1) The models selected are often larger than required for the new task at hand as the original intent of the model was to achieve state-of-the-art performance on large competition datasets and 2) Transfer learning works best when the new domain resembles that of the domain the base model was trained on. This research is therefore also motivated to embed human knowledge within automated techniques of designing new and novel CNN model architectures as an alternative to transfer learning and manual trial and error.

1.3.1 Motivation and Research Questions for Efficiency Focused CNN Architectures

Designing CNN models for resource constrained environments is an important research topic. The reason this is an important area is that many applications for computer vision exist on IoT devices. Examples of applications include people detection on CCTV cameras and face detection on smartphones, both of which lack powerful graphics cards. Deep CNN models are slower to train and run inference on [4]. In industry and academia, the development and application of a CNN model into a resource contained environment such as a CCTV camera, IoT device or robotics are constrained in terms of the available resources. In such a contained environment, a deep CNN model may simply run too slow for the application at hand. The motivation of this research is to address the following research problem by proposing a novel CNN architecture better suited for resource contained environments by answering the following research question:

- Is there a better way to construct a CNN model so that the model achieves greater accuracy levels with less computational cost, resulting in more accurate CNN models that can be deployed into resource contained environments such as IoT devices?

1.3.2 Motivation and Research Questions for Automating CNN Architecture Generation

Automatically designing CNN models is an important research topic because designing such models by hand requires specialist knowledge and trial and error. The motivation of this research is to design an automatic procedure for deep CNN model generation. The research problems addressed in this study include:

- Can CNN model architectures be encoded in a way that it is ensured to be both architecturally valid and reasonably built to avoid the need for additional hard coded governing rules [10] or wasteful function evaluations?
- Can each particle be guided effectively through a complex search space efficiently in order to construct more effective networks, that are faster than the current state-of-the-art algorithms [10] [11]?
- Can CNN models be designed to be more efficient using an algorithm that is both easy to understand and fast to run, so that the approach can be easily exploited in both academia and industry settings with limited specialised knowledge while not compromising the overall performance?

1.3.3 Motivation and Research Questions for Automating Residual CNN Architecture Generation

Automatically designing deep residual CNN models is an important research topic because some computer vision applications are more suited to deeper architectures owing to the challenging nature of the classification task at hand. No existing works perform the automatic design of residual deep neural networks, so the motivation of this research is to design an automatic procedure for deep residual CNN model generation. The research problems addressed in this study include:

- Can a rich CNN model architecture search be performed whilst exploiting advanced techniques such as residual connections so that larger models can be constructed?
- How can a search be performed that is not susceptible to becoming trapped in local optima?

1.4 Contributions

The following contributions are related to a field of research aimed at designing convolutional neural networks which are more accurate and efficient. Contributions in Section 1.4.1

relate to the manual design of a more efficient model with greater accuracy than existing efficiency-focused models. The contributions in Sections 1.4.2 and 1.4.3 relate to embedding human knowledge within automated techniques for designing accurate CNN model architectures specifically for the task at hand, as opposed to applying transfer learning techniques using a large base model. The hypothesis is that a model constructed specifically for the task at hand will be no larger than required, and the models maximum size could be contained by search parameters to suit the target environment. This field of work is important as edge deployments such as models deployed on IoT devices such as CCTV cameras have limited hardware resources. Specifically, they lack graphics cards that are used to accelerate the inference speed of a deep CNN model and instead rely on comparatively slower CPU processing. The current works in this field relate to constructing model architectures that contain network elements designed to reduce the number of floating point operations to speed up inference, and the automated design of CNN model, designed to perform well on a target data set. The contributions in Section 1.4.1 add to this field by proposing an efficiently focused CNN model which is more accurate and efficient than the current state-of-the-art approaches. Furthermore, the contributions in Sections 1.4.2 and 1.4.3 add to the field by proposing automated techniques that embed human knowledge for designing deep CNN models. Automated techniques add to the field as they provide an alternative to transferring learning. Transfer learning techniques involve selecting an existing model as a base model and subsequently train some of its final layers on a new data set. Because the base model was not designed for the new data set and instead commonly designed to achieve state-of-the-art accuracy on competition data sets, designing a model through transfer learning does not ensure optimal model design for the new task at hand. The proposed automated techniques include search bounds to limit the proposed models depth and width so that the final model could suit a resource-contained environment and construct a model specifically suited for the task at hand.

1.4.1 Contributions for Efficiency Focused CNN Architectures

The main contributions of this research are as follows.

- Proposing a new architecture, namely IoTNet, which is designed specifically for performance constrained environments such as IoT devices, smartphones or embedded systems. It trades accuracy with a reduction in computational cost differently from existing methods by employing novel pairs of 1x3 and 3x1 normal convolutions, rather than using depth-wise separable convolutions.

- An in-depth comparison of the proposed architecture against efficiency-focused models including MobileNet [5], MobileNetV2 [7], ShuffleNet [6] and EffNet [9] has been conducted using CIFAR-10 [18], Street View House Numbers (SVHN) [44] and German Traffic Sign Recognition Benchmark (GTSRB) [45] data sets. The empirical results indicate that the proposed block architecture constructed exclusively from pairs of 1x3 and 3x1 normal convolutions, with average pooling for downsampling, outperforms the current state-of-the-art depth-wise separable convolution-based architectures in terms of accuracy and cost.

- A direct comparison of pairs of 1x3 and 3x1 normal convolutions against 3x3 standard convolutions has also been conducted. The empirical results indicate that the proposed approach results in a more accurate and efficient architecture than a scaled-down large state-of-the-art network.

1.4.2 Contributions for Automating CNN Architecture Generation

This research has the following contributions, which aim to address the aforementioned research problems.

- A new group-based encoding strategy is proposed. Each group contains at least one convolutional layer. Its final layer is reserved as an optional pooling layer. The number of groups can be adjusted in accordance with the input image size. By restricting the number of groups, it can adapt the frequency of the pooling operations toward the input image size. As such, it ascertains the position and maximum frequency of the pooling operations always result in a valid model architecture without the need for additional governing rules.

- A new velocity updating mechanism based on the key network configuration differences is developed. Existing models such as psoCNN [10] copy the layers randomly from the global and personal best solutions for architecture generation. This indicates that new models are always generated based on the combinations of existing layer configurations. To overcome such limitations, the new velocity updating mechanism creates new network architectures by identifying the key layer configuration differences between particles. This proposed mechanism is capable of devising new network architectures by exploring the intermediate positions of the particles' trajectories. It is also less dependent on the requirement of a good random swarm initialization.

- A new position updating mechanism with weighted velocity strengths is devised. This granular position updating mechanism enables a thorough exploration of the search space and increases the likelihood of generating diversified network configurations. It employs a weighted strength of the velocity updates for new position generation, leading to the exploration of the search space in various forces and scales to increase the chances of formulating diversified networks. Such a granular movement also enables a better balance between intensification and diversification in order to increase the chances of finding global optimality. Both proposed encoding and search strategies illustrate significant capabilities in enhancing performance and computational efficiency.

- A comprehensive evaluation of the proposed model with several data sets is conducted. Our proposed model compares favourably with the state-of-the-art models such as psoCNN [10] and notable alternative methods including EvoCNN [12]. Serving as a practical alternative to deep network generation, the proposed model achieves up to 7.58% improvement in accuracy and up to 63% reduction in computational cost, in comparison with those from the current state-of-the-art methods. Importantly, the proposed model is repeatable and easy to implement with limited hardware resources.

1.4.3 Contributions for Automating Residual CNN Architecture Generation

Specifically, the contributions of this research are as follows.

- A novel PSO algorithm, namely resPsoCnn, is proposed for residual deep architecture generation. First, a new group-based encoding scheme is proposed, which provides compatibility with residual connections. Each group contains one or more residual convolutional blocks and an optional pooling layer. Each residual block inside a group shares the same number of filters so that the residual operation can be performed. The number of filters per group is optimised to control the network width. Before each group, a transitional layer is used to increase or decrease the number of filters to the number required by the subsequent group. The kernel sizes of convolutional layers are individually encoded, giving fine-grained control over the receptive field of each block. The number of blocks within each group can vary in order to increase or decrease the model depth, while different pooling layer types are embedded to control downsampling.

- Proposing an optimization strategy that exploits the advantages of residual connections to avoid the vanishing gradient problem. Such a strategy addresses weaknesses in related studies which either 1) perform optimization tasks only on fixed skeleton models (e.g. fixed numbers of blocks) that make use of residual connections but restrict diversity as settings such as kernel sizes and pooling types are fixed, or 2) do not use residual connections and instead optimize a range of hyperparameter settings, capable of producing diverse, but shallow networks. Our proposed strategy addresses both of the aforementioned weaknesses by providing the ability to make use of residual connections to construct deep network architectures whilst also optimizing a range of network settings to improve diversity.

- Proposing a new velocity updating mechanism that adds randomness to the updating of the group and block hyperparameters. Specifically, it employs multiple elite signals, i.e. the swarm leader and the non-uniformly randomly selected neighbouring best solutions, for searching optimal hyper-parameters. The hyperparameter updating at the group and block levels is conducted by either selecting from the distance between the current particle and the global best solution, or the distance between the current particle and a neighbouring best solution, to increase search diversity. The proposed search mechanism optimizes the number of groups, network width and depth, kernel sizes, and pooling layer choices to produce a rich assortment of candidate best solutions of residual deep architectures. Owing to the guidance of multiple elite signals (e.g. the global best and neighbouring promising solutions), the proposed search process achieves a better balance between exploration and exploitation and addresses a weakness in existing search methods where the search processes led by the single leader are prone to being trapped in local optima and converge prematurely. Evaluated using a number of benchmark data sets, the devised networks show superior performances against those yielded by several state-of-the-art existing methods.

1.5 Thesis Layout

Section 2 introduces the main concepts relating to Convolutional Neural Networks (CNNs). This section reviews related studies focused on CNN models that aim to maximise model accuracy. CNN models that are focused on efficiency with respect to computational cost are reviewed in this section. The various applications for CNN models have been reviewed. Moreover, this section reviews evolutionary algorithm based approaches for automatically constructing and optimizing CNN models.

Section 3 introduces a novel efficiency focused CNN model, namely IoTNet. The proposed CNN model saves computational cost by factorizing a 3x3 convolution into a pair of 1x3 and 3x1 convolutions. The proposed model is compared with efficiency focused CNN models and scaled down versions of accuracy focused CNN models in respect to accuracy and computational cost across a range of data sets. The proposed CNN model introduced in this section has been published in the following journal publication [46].

Section 4 introduces a PSO based CNN architecture generation approach. The novelties introduced include an encoding strategy capable of encoding deep CNNs, and a novel particle search strategy. This section provides the results of the proposed PSO architecture generation approach, compared against state-of-the-art related work. The proposed PSO based CNN architecture generation approach introduced in this section has been published in the following journal publication [47].

Section 5 introduces a PSO based CNN architecture generation approach for CNNs with residual connections. The novelties introduced include are as follows. 1) An encoding strategy capable of encoding deep CNNs. The CNN models constructed contain residual connections. The residual connections enable deeper networks to get constructed, and 2) a novel particle search strategy that follows the neighbouring and global best solutions to improve search capabilities. This section provides the results of the proposed PSO architecture generation approach, compared against state-of-the-art related work. The proposed PSO based residual CNN architecture generation approach introduced in this section has been published in the following journal publication [48].

Section 6 summarises the main findings and results from the conducted studies. This section discusses the future direction of the research. The application of this research in both academia and industry over the next 3 to 5 years is discussed.

2 Related Works

2.1 CNN Concepts

Prior to convolutional neural networks (CNNs), pattern recognition tasks were performed by devising systems that comprised of a feature extractor and a classifier. The feature extractor was responsible for transforming high dimensional data into a series of low dimensional vectors. These low dimensional vectors were designed for a specific domain by hand so that pattern matching could be performed. The weakness of this approach was that the success depended on the ability of the designer to manually construct an appropriate feature extractor. The second module comprised of a trainable classifier such as a support vector machine [49], which was limited to the classification of low dimensional vectors only, due to available hardware at the time. This limitation meant that such a module could only classifying easily spreadable classes.

With the advent of more powerful and low cost hardware capable of performing mathematical calculations quickly, along with the availability of larger quantities of training data, it became possible to train models such as neural networks via gradient based backpropagation, popularized by [50]. The weakness of using a neural network for performing image classification tasks was that the first layer of such a neural network must flatten an image into a 1 dimensional vector. As images are typically large, such a network would require a vast memory footprint to store weight information relating to the connections between the first layer, and subsequent hidden layers. Another weakness of neural networks performing image classification is that as the image is flattened into a 1 dimensional vector, it becomes unstructured. For example, two images of a handwritten digit of the same character will contain similar features, but as the features are not organised during the flattening process, this leads to intolerance in respect to image translations.

Convolutional Neural Network (CNNs) were proposed to solve the weaknesses of neural networks when performing classification tasks on complex data such as image. CNNs are a specialised neural network architecture first introduced by LetNet-5 [1] to address the aforementioned weaknesses of neural networks within the domain of image classification. An overview of LeNet-5 is shown in Figure 1.

The architecture of LeNet-5 comprises of layers. The layer types are convolutional layers, subsampling layers and fully connected layers.

Convolutional layers are feature extractors. Within the first convolutional layers of a CNN model, such layers extract low level features such as edges. As these features propagate through the network, they are combined by subsequent layers to form higher level features. The input of a convolutional layer is a volume of shape Df^2M where Df represents the width and height, and M represents the number of channels. The first layer of LeNet-5 is indicated in Figure 1 as C1. Layer C1 takes as input a greyscale image of shape 32x32x1. A convolutional layer contains N kernels k where N is a design choice that increases the number of channels of the output. Increasing N is useful for extracting more features from

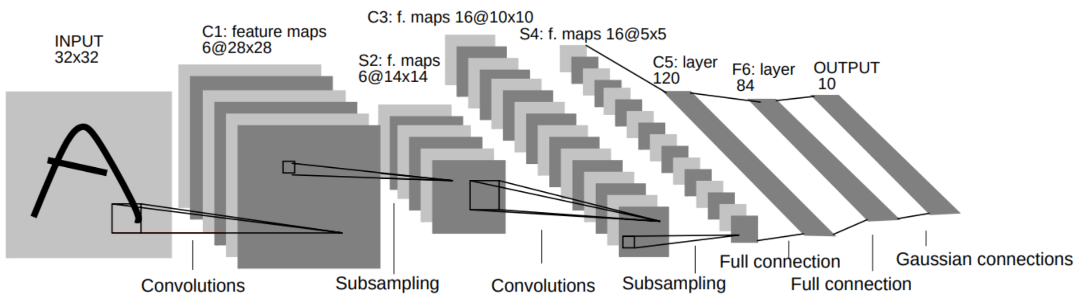


Figure 1: LeNet-5 [1] comprises of convolutional layers indicated as C for local feature extraction, subsampling layers indicated as S to reduce layer dimensions and fully connected layers indicated as F for performing classification.

the same regions of the input volume. In the case of layer C1, $N = 6$.

A kernel is a matrix, which in the case of C1 is of size $5x5$, therefore each kernel contains 25 trainable weight parameters, plus a trainable bias. The convolution operation is performed by first positioning the kernel in the top left corner of the input volume. The kernel is then moved across the entire input volume with a stride of 1. For every position that the kernel is placed, a dot product is taken that results in an output volume of 28x28x1. The convolutional operation is performed with each k kernel, which in the case of layer C1 is 6 kernels. The output of each convolutional operation for all 6 kernels are then stacked to produce a final output of shape 28x28x6.

The convolutional operation means that each 5x5 region of the input volume is directly connected to all N kernels. This is referred to as the receptive field. The larger the kernel dimensions, the larger the region which is in focus for each dot product operation. The advantage of a convolutional layer over a fully connected layer of a neural network is that the location of a detected feature is of less importance. The total number N of kernels k controls the models width. Increasing the number of kernels means that more features can get extracted from the same region of the input volume. Increasing N however increases the number of trainable weights, and therefore increases the computational cost and memory footprint of the model.

Sub-sampling layers within LeNet comprise of a 2x2 kernel which is slid across the input volume with a stride of 2. The kernel of a sub-sampling layer contains trainable coefficients. Due to the fact that the kernel size is 2, and the size is 2 means that the output volume is half of that of the input. The sub-sampling layer has two main aims, firstly features from the input are summarised into higher level features, and secondly, as the output volume is reduced, so are the number of trainable parameters which improves computational efficiency. More recently, sub-sampling has been replaced with pooling layers which rather than performing dot operations with trainable coefficients, instead take either the average or mean for each kernel position.

Fully connected layers are used to flatten the output and reduce its dimensions to that of the total number of classes. The purpose of this is to perform classification. Activation functions add non linearity. Sigmoid activation functions which are also referred to as logistic function or squashing functions are popular for binary classification tasks. For multi-class classification tasks, softmax [51] is a populate choice. For example, ResNet [3] used a 1000-way fully-connected layer with softmax activation function to classify the COCO data set [17].

2.2 Accuracy Focused CNN Architecture

Research into architectures that improve the accuracy and performance of CNNs has been an active research area for some time. This work has resulted in architectures such as AlexNet [2]. The goal of AlexNet was to propose a model capable of classifying ImageNet [52], a challenging data set containing 1000 different classes. In comparison to LeNet-5, AlexNet contained 8 layers rather than 5. Each layer had more kernels than LeNet-5 to improve the learning capacity of the model. The authors adopted a larger kernel size i.e an 11x11 kernel for the first layer, to increase the receptive field size. AlexNet contained approximately 60 million parameters. To train such a large model on the limited hardware available at the time, the authors split the model into two parts so that it could be trained on two graphics cards, as indicated in Fig 2. AlexNet was the state of the art model on ImageNet in 2012. The weakness of AlexNet was the model complexity which meant it was hard to make deeper as it would become impractical to train.

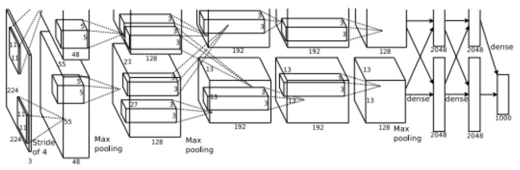


Figure 2: AlexNet was a complex model split training over two graphics cards by splitting the model into two parts. The top part of the model was trained on graphics card 1, and the bottom part was trained on graphics card 2. [2]

Works such as VGG-16 [53] set out to address the model complexity weakness of AlexNet. The authors fixed all kernel sizes to 3x3. The rationale for fixing the kernel was to reduce the number of floating-point operations needed to perform a convolution. The reduction in floating-point operations meant that VGG-16 could contain 16 layers. VGG-16 was therefore twice as deep as AlexNet. A weakness of VGG-16 was that it was deeper and therefore susceptible to the vanishing gradient problems [3].

Further studies have shown that model depth can improve overall accuracy [54] [55] [56], but to achieve greater depth, the vanishing gradient problem needed solving. Works such as ResNet [3] set out to address the vanishing gradient problem by introducing residual connections. The authors organised network architectures into a series of blocks. Each block contains two convolutional layers. The input to the block is fed through the convolutional layers inside the block to produce an output. A residual connection is defined as taking the blocks output then summing it with the block input x as shown in Fig 3 as $F(x) + x$.

The benefit of a residual connection is that it allows for gradients during backpropagation to propagate further, as they do not need to pass through as many convolutional layers.

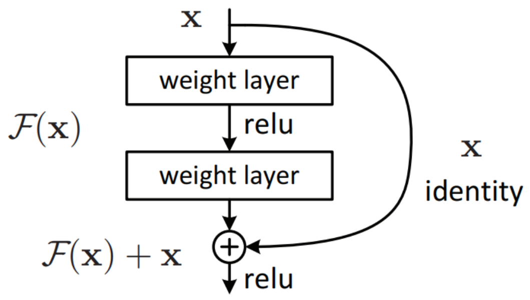


Figure 3: A ResNet block [3] feeds an input through a pair of convolutional layers, before summing the output with the input.

Studies such as WideResnet [4] aimed to address a weakness in Resnet, stating that Resnet models are capable of containing thousands of layers, but increasing the depth further results in minor improvements. The authors set out to show that it is important to balance the width of a model i.e the number of kernels used per convolutional operation, and not just increase its depth. The authors extended the ResNet block by adding a dropout layer [57] to prevent overfitting as shown in Fig 4.

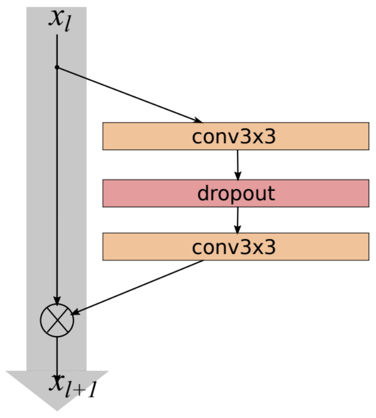


Figure 4: A WideResnet block [4] contains a pair of convolutional layers, a dropout layer and a residual connection.

The model architecture was organised into three groups. Each group contained N WideResNet blocks. A scaling factor k was introduced to control the width of each group. The width of the first group was $16k$, the second group had a width of $32k$, and the third group had a width of $64k$, as indicated in Fig 5.

Experiments were conducted on a range of different values for N and k. The best performing model on the CIFAR-10 [18] data set had 28 layers, and a wide construction i.e $k = 28$. The proposed shallower and wider model outperformed the deeper, thinner model i.e 40 layers deep and $k = 8$. The shallower but wider configuration was reported as being 0.49% more accurate than the aforementioned deeper counterpart.

Other related works which address the vanishing gradient problem include DenseNet [58]. The authors of Densenet found that deep convolutional networks could be constructed

group name	output size	block type = $B(3,3)$
conv1	32×32	[3×3, 16]
conv2	32×32	$\begin{bmatrix} 3\times3, 16\times k \\ 3\times3, 16\times k \end{bmatrix} \times N$
conv3	16×16	$\begin{bmatrix} 3\times3, 32\times k \\ 3\times3, 32\times k \end{bmatrix} \times N$
conv4	8×8	$\begin{bmatrix} 3\times3, 64\times k \\ 3\times3, 64\times k \end{bmatrix} \times N$
avg-pool	1×1	$[8 \times 8]$

Figure 5: A WideResnet model is organised into groups. Each group contains N blocks. Each blocks width is controlled by a scaling factor k [4]

by concatenating the inputs to all previous layers, with subsequent layers. The weakness of this approach is that the model is a very domain-specific model and prone to overfitting due to the high number of layer interconnections [59].

2.3 Efficiency Focused CNN Architectures

Works relating to IoT devices identify a real need for more accurate models within resource-constrained environments. Recently, the research studies of [60] highlighted how IoT devices, such as Raspberry Pi, make edge computing a reality, as cheap devices can be interconnected to form network infrastructures. When interconnected, such networks have been used to tackle a range of problems, including pollution, air, water, food, and fire sensing, heartbeat and blood pressure monitoring, and motion tracking. Furthermore, the studies of [61] presented a novel solution to decentralize data exchange based on wireless IoT and blockchain technologies and highlighted how IoT-based solutions have illustrated exponential growth owing to a rise in IoT applications within smart cities and health tracking domains. Because of the fast growth and the range of interesting applications for IoT devices, more accurate and efficient deep learning models are essential. Moreover, a recent case study [62] especially focused on sensor reliability relating to LiDAR sensors with IoT capabilities. Their work pointed out that such types of sensor devices are becoming widespread. Their IoT capable devices employed a range of models to perform tasks such as driver assisting obstacle detection within cars and fault detection, yet more advanced deep learning models could be deployed in such applications providing such networks can deliver sufficient accuracy and efficiency.

Other research efforts in building network architectures suitable for use on performance restricted environments such as IoT and smartphones have led to another category of models, specifically designed to be computationally efficient. State-of-the-art architectures of these types of models include MobileNet [5], MobileNetV2 [7], ShuffleNet [6], LiteNet [8] and EffNet [9].

A comparative analysis of all related studies has been summarized in Table 1, followed by in-depth discussions.

The motivation behind MobileNet [5] illustrated in Figure 6 was to reduce the network computational cost by using 3x3 depth-wise separable convolutions. Specifically, a depth-wise separable convolution is a form of factorization which reduces computational cost in comparison with a standard convolution. A more comprehensive comparison between a normal convolution and a depth-wise separable convolution is provided in Section 3.2. The study in [5] showed a drawback when evaluated with ImageNet, i.e., the performance of MobileNet decreased by 1%, but the advantage was a substantial reduction in computational cost in comparison with those of the model with a normal 3x3 convolution.

The 3x3 convolution has proven a popular choice for many architectures but Inception-v3 [63] and Inception-v4 [64] have shown that the 3x3 convolution can be replaced with a 3x1 and 1x3 convolution, resulting in a 33% reduction in parameters. While the above variants of the inception block make use of 1x3 and 3x1 convolutions, the block contains multiple branches, filter concatenations and 1x1 convolutions. Multiple branches were proposed within the inception model to train deeper models. The drawback of such a practice is that in resource-constrained environments, models tend to be shallower due to the computational constraints and multiple branches substantially increase the computational cost for a given depth. In comparison with these existing models, the proposed architecture in this research differs from inception networks as it contains one branch, and has no filter concatenation which reduces overhead and does not use 1x1 convolutions.

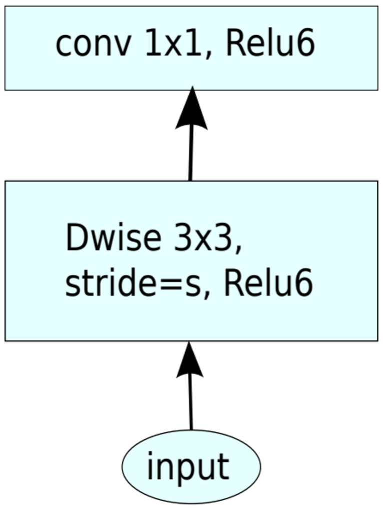


Figure 6: MobileNet [5] uses depth-wise separable convolutions. DWise denotes depth-wise convolution. Residual connections are not used.

ShuffleNet [6], as illustrated in Figure 7, uses two new operations, i.e., a point-wise group convolution and channel shuffling. A 3x3 kernel was used for the depth-wise portion of the depth-wise separable convolution operation to reduce computational cost. The motivation of using a shuffle was to combat pitfalls in group convolutions. Specifically, if multiple group convolutions stack together, output channels are only derived from a small fraction of input channels which impacted performance. Shuffling the channels overcame this problem and led to performance improvements over MobileNet. However, this additional operation, i.e., shuffling, is also a drawback, as it leads to additional computation.

MobileNetV2 [7], as illustrated in Figure 8, builds on the original MobileNet architecture

Table 1: Comparative analysis of related studies.

Model	Kernel	Convolution Type	Emphasis	Methodologies and Strengths	Drawbacks
AlexNet [2]	mixed	standard	accuracy	Demonstrated how the model depth was essential for performance	Contained large kernels which are less efficient. Outperformed by subsequent studies
ResNet [3]	3x3	standard	accuracy	Used residual connections to enable training deeper networks	A slim but deep state-of-the-art model, not designed for constrained environments
Inception [63, 64]	mixed	standard	accuracy	Trained deeper networks using sparsely connected network architectures, i.e., by using a variety of kernel sizes side by side	The employed side-by-side model increased model complexity
WideResnet [4]	3x3	standard	accuracy	Demonstrated that widening a residual network can decrease its depth and improve its performance	A state-of-the-art model, not designed for constrained environments. Less efficient at smaller scales than the proposed approach
PyramidNet [56]	3x3	standard	accuracy	Gradually increasing the feature map size of deep networks led to performance improvements on ResNet	A deep state-of-the-art model, not designed for constrained environments. Gradual depth increase led to a larger model size
MobileNet [5, 7]	3x3	depth-wise	efficiency	Traded accuracy with efficiency by using depth-wise separable convolutions	Contained bottlenecks during downsampling which impeded data flow
ShuffleNet [6]	3x3	depth-wise	efficiency	Shuffling channels helped information flowing when performing depth-wise separable convolutions	Shuffle resulted in additional operations and contained bottlenecks which impeded data flow
EffNet [9]	1x3 and 3x1	depth-wise	efficiency	Factorized 3x3 depth-wise convolutions into 1x3 and 3x1 depth-wise convolutions to reduce complexity. Addressed bottlenecks of prior efficiency-focused models	Based on depth-wise separable convolutions which traded accuracy with efficiency less optimally than the proposed approach
LiteNet [8]	1x2 and 1x3	depth-wise and standard	efficiency	Combined ideas from Inception and MobileNet	A combination of drawbacks of Inception and MobileNet (see above)
Ours	1x3 and 3x1	standard	efficiency	Factorized 3x3 into 1x3 and 3x1 standard convolutions to retain the strength of standard convolutions, i.e., superior performance while reducing model complexity	Designed for constrained environments and not to outperform state-of-the-art accuracy-focused models in extremely large configurations on GPU machines.

using 3x3 depth-wise separable convolutions but with the addition of an inverted residual structure where shortcut connections are used between thin bottleneck layers to reduce input and output sizes. This model outperformed the state-of-the-art networks such as MobileNet and ShuffleNet, at the time for the evaluation of ImageNet [52].

LiteNet [8], as illustrated in Figure 9, takes an inception block which contains 1x1, 1x2 and 1x3 standard convolutions arranged side by side and makes modifications (inspired by MobileNet) by replacing half of the 1x2 and 1x3 standard convolutions with their depth-wise equivalents. Their proposed block, therefore, contains a mix of both standard and depth-wise separable convolutions. Their work also makes use of a SqueezeNet fire block [65] to further reduce the total network parameters. The model was trained on the MIT-BIH electrocardiogram (ECG) arrhythmia database [66] and improved the accuracy rate against baseline models of $\approx$0.5%. The drawback of their proposed model is the side-by-side structure employed, since side-by-side blocks increase the total number of parameters for a given depth. The inception model originally proposed a side-by-side block to reduce the need to select appropriate filter sizes upfront. By including a variety of different filter sizes side by side, the network could learn which ones are best to use. The research community has since learnt in related works that the most common filter used is 3x3 and deeper models perform better.

A common drawback of MobileNet, MobileNetV2 and ShuffleNet is a substantial re-

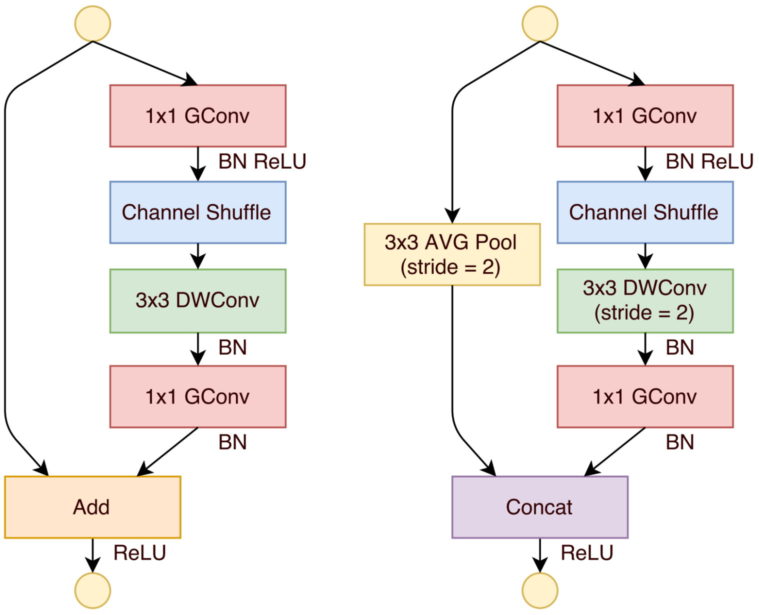


Figure 7: ShuffleNet [6] uses a 3x3 convolution for the depth-wise phase of the convolution which is performed after a channel shuffle. DWConv denotes depth-wise convolution. This architecture uses residual connections.

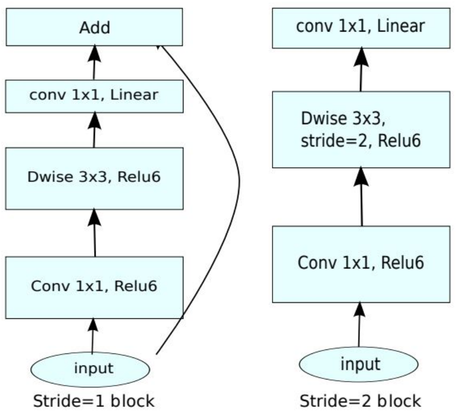


Figure 8: MobileNetV2 [7] uses a 3x3 convolution for the depth-wise phase of the convolution and makes use of residual connections. DWise indicates depth-wise convolution.

duction in the total number of floats-out when downsampling is performed. The authors of EffNet [9] highlighted this as a weakness as the aggressive nature of the reduction is that floats cause a bottleneck which impedes data flow when the model is small, causing them to diverge. The motivation for EffNet as illustrated in Figure 10 was to deploy networks in performance constrained environments and to increase the efficiency of existing off-shelf models. EffNet achieves this by gradually reducing the total number of FLOP outputs throughout the network to avoid bottlenecks. EffNet also replaced 3x3 convolutions with pairs of 1x3 and 3x1 convolutions performed as a depth-wise separable operation to further reduce computational cost. A weakness to such an approach is that the computational saving of performing a 1x3 convolution as a depth-wise operation is less than that of a 3x3 convolution as elaborated in Section 3.2.

Besides the above methods, post-processing techniques exist which reduce model com-

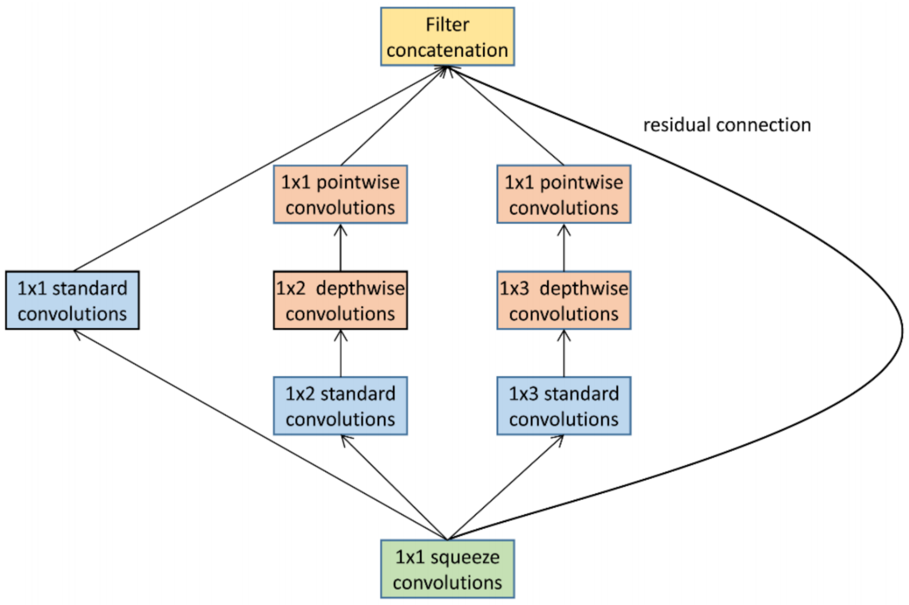

Filter concatenation
residual connection
1x1 pointwise convolutions
1x1 pointwise convolutions
1x1 standard convolutions
1x2 depthwise convolutions
1x3 depthwise convolutions
1x2 standard convolutions
1x3 standard convolutions
1x1 squeeze convolutions

Figure 9: LiteNet [8] takes an inception block and replaces one of the 1x2 convolutions and one of the 1x3 convolutions with their depth-wise counterparts, respectively.

plexity and therefore the computational cost. Related studies in this field include [67, 68, 69, 70], which employed pruning algorithms that remove redundant elements such as filters with do not assist in the final classification from a model as part of a post-processing operation. These developments indicated that a model can be compressed to reduce complexity, with minimal impact on performance. Related works such as [67] proposed a pruning algorithm based on Taylor series expansion of a cost function which was applied to SqueezeNet [65], resulting in a 67% model reduction. Some limitations of this approach include a 1% drop in accuracy. It obtains better results when training from scratch, rather than using transfer learning on top of a pre-trained network. Studies such as [68] prunes based on a filter stability which is calculated during training. As an example, unstable filters are candidates for pruning. This approach was applied to LeNet-5 [71] on MNIST [72], VGG-16 [73] on CIFAR-10 [18], ResNet-50 [3] on ImageNet [52], and Faster R-CNN [74] on COCO [17] and reduced the number of FLOPs by a factor of 6.03X. A limitation to this approach is that it can only be used on new models trained from scratch.

In contrast to post-processing techniques, architecture generation algorithms such as [75, 76, 13, 77, 78] have demonstrated that architectures can be automatically generated by exploring different architecture choices and hyper-parameter settings. Studies such as [76] used a Q-Learning method [79] with an epsilon-greedy exploration strategy [80] to speed up the time taken when generating new model architectures. The algorithm was able to choose from 1x1, 3x3 or 7x7 convolutions and was trained on CIFAR-10. The approach was able to reduce the time required to generate suitable architectures from 22 days for the current state-of-the-art approach [11] to 3 days with a 0.1% reduction in the error rate. Studies such as [75] recently proposed an ageing evolution algorithm which extended the well-established tournament selection in genetic algorithm [81] by introducing an age property to favor younger

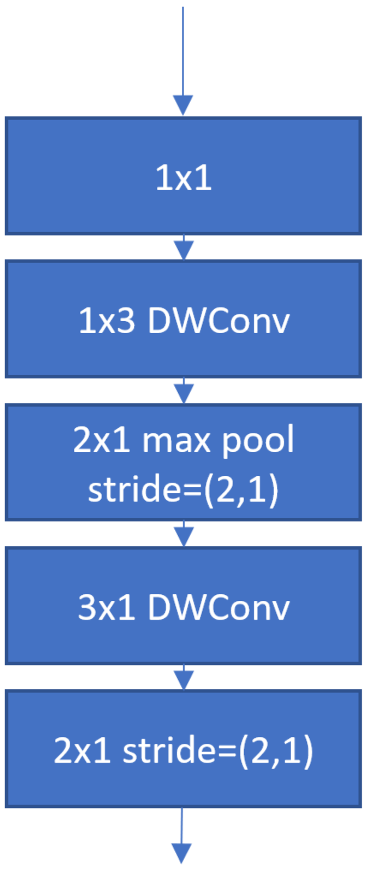


Figure 10: EffNet [9] uses 1x3 and 3x1 depth-wise separable convolutions to reduce model complexity. DWConv denotes depth-wise convolution.

genotypes. The algorithm chose from 3x3, 5x5 or 7x7 separable convolutions, 1x7 then 7x1 standard convolutions, 3x3 max or average pooling and dilated convolutions. The approach achieved a new state-of-the-art 96.6% top-5 accuracy rate on ImageNet. These evolving model generation methods require additional computational resources owing to the large search space and complex evolving processes with the involvement of fitness evaluations.

Parameter quantization is an area of research which aims to make a network have a smaller memory footprint by compressing 32-bit parameters to 16-bit or even smaller. Related developments such as [82, 83, 84] have explored compression to various degrees, e.g., including reducing weights to binary values. Bi-Real Net [82] significantly reduced memory footprint and computational cost by setting all weights and activations to binary values. This process was achieved by using a sign function which replaced the true activations and weights with either -1 or 1. It also reduced the memory usage of the previous state-of-the-art 1-bit CNN XNOR-Net [85] by 16 times and reduced computational cost by 19 times. The authors of [84] introduced chunk-based accumulation and floating-point stochastic rounding functions which compressed weights from 32-bit to 8-bit. In comparison with a wide spectrum of popular CNNs, for the evaluation of several benchmark data sets, their network achieved similar accuracy rates as those of the baseline models, but with reduced computational costs. However, the study also indicated that their model suffered from loss of precision over the 32-bit model counterparts.

Learning data augmentation strategies which can be transferred across different data sets and models such as [86] have proved extremely effective at improving model accuracy by discovering novel combinations of data augmentations which can be applied to specific data sets and often transferred to others.

The above studies on pruning algorithms, automatic architecture generation and parameter quantization are examples of related work, which could complement ours and be embedded for future development.

2.4 Application of CNN Models

There are many recent applications of CNN models. For image classification tasks, the authors of [87] reduced the analyzing time and increased the accuracy of medical diagnostic by training a LeNet [1] convolutional neural network to distinguish between Staphylococcus and Lactobacillus bacteria.

Pre-trained CNN models such as AlexNet [2] have been further trained through transfer learning to perform the detection and classification of lung abnormalities in related studies such as [88]. The authors showed that with a relatively small data set i.e 163 CT scans of patients with lung nodule cases, and 372 CT scans of patients with or without diffuse lung abnormalities, the trained model was able to achieve a 95.2% classification accuracy. Furthermore, the authors were able to improve the accuracy to 99.4% by applying rotation and reflection data augmentation. The rotation and reflection data augmentations increased the amount of training data 8 times.

The authors of [89] constructed a convolutional neural network to perform breast cancer image classification. The CNN model consisted of a stack of 3 convolution layers, a ReLU activation layer, followed by max-pooling layer. The image data was 8-bit greyscale images. The authors proposed extracting the bits from each image pixel into 8 images through a process called bit-plane slicing. Bit-plane slicing involved extracting all the pixels with the value of 1 into a new image representing bit-plane one. This process was repeated for all eight possible pixel values, resulting in eight different bit-planes. The authors reported that the fourth, fifth, sixth and seventh bit-plane images had a higher classification accuracy than with the original image. The weakness of this work is that deeper models were not constructed to see if the classification accuracy improved with the original images.

Works such as [90] [91] proposed a CNN model for performing oil spill detection from Space-borne Synthetic Aperture Radar(SAR) images. [92] applied transfer learning to a VGG16 [73] model to distinguish between normal and pneumonia cases from chest X-ray images. [93] proposed a diverse, multi-branch CNN model to perform hyperspectral image classification, within the domain of remote sensing. CNN models are also applied in manufacturing settings. [94] proposed a CNN model for performing fault diagnosis of rolling bearing, achieving an impressive 99.2% accuracy at identifying bearing faults. Other applications of CNN models include detecting material defects in industrial settings [95] and addressing medical problems such as skin lesion segmentation [96], fall detection [97] and health monitoring [98] [99].

CNN models have also shown groundbreaking results in classifying handwritten digits [1]. Object localization models e.g. YOLO [100] [101], Fast R-CNN [102] and Faster R-CNN [74] and object segmentation methods e.g. Mask R-CNN [103] all make use of CNNs as the backbones to perform detection, localization and segmentation tasks. Advances in automated CNN architecture generation, therefore, have a significant impact in many domains.

In summary, CNN models can broadly be split into accuracy focused models, and efficiency focused CNN models. Accuracy focused CNN models primarily aim to maximize accuracy on a particular data set. They contain strategies that facilitate the construction of large networks to help improve accuracy. Accuracy focused models are often deep networks that run on powerful graphics card devices. For example, ResNet was trained on two GPU machines [3]. Efficiency focused models focus on strategies that reduce the number of floating point operations (FLOPs) as doing so means fewer computational steps are required to perform one forward pass through the network. This means that such a model can be run in more resource constrained environments. The main strategy to reduce computation cost is to perform depth-wise separable convolutions instead of standard convolutions. Depth-wise separable convolutions are effective at reducing computational cost, at the expense of accuracy [5]. CNN models in general, have a broad range of applications and positively impact many domains. Domains that benefit from the application of CNN models include image classification, object detection, medical diagnosis, mechanical fault diagnosis, health monitoring and remote sensing. The broad range of applications for CNN models indicates that the research conducted in the following chapters should have a wide-reaching and positive impact.

2.5 Evolutionary Algorithms

2.5.1 PSO Concepts

Proposed by [32], PSO is a popular swarm intelligence algorithm which simulates natural social behaviours, e.g. bird flocking or fish schooling. The concept of PSO is to create a swarm of particles, where each particle explores a search space guided by the best-known position of the entire swarm, g_{best}, as well as its individual best experience, p_{best}. In each iteration, the particle position is updated by adding a velocity to the current position vector. The formula for velocity calculation can be divided into three main components, i.e. inertia, cognitive, and social components.

The inertia component shown in Equation 2 multiplies the current velocity V for particle i in the t-th iteration with a weight w, which controls the impact of the previous velocity on the new velocity calculation.

$$inertia = wV_i^t \tag{2}$$

The cognitive component shown in Equation 3 multiplies the distance between the current particle position X and its personal best solution P by a cognitive acceleration coefficient c_1 as well as a random parameter r_1.

$$cognitive = c_1 r_1 (P_i^t - X_i^t) \tag{3}$$

Similarly, the social component shown in Equation 4 multiplies the distance between the current particle position X and the global best solution G by a social acceleration coefficient c_2 and a random value r_2.

$$social = c_2 r_2 (G^t - X_i^t) \tag{4}$$

The acceleration coefficients c_1 and c_2 control the degrees at which a given position update is guided by the cognitive or social component. The complete velocity updating formula shown in Equation 5 produces the final velocity for the $(t+1)$-th iteration.

$$V_i^{t+1} = wV_i^t + c_1 r_1 (P_i^t - X_i^t) + c_2 r_2 (G^t - X_i^t) \tag{5}$$

The new position in the $(t+1)$-th iteration is produced using Equation 6, based on the velocity yielded from Equation 5.

$$X_i^{t+1} = X_i^t + V_i^{t+1} \tag{6}$$

2.5.2 Other Evolutionary Algorithms

Firefly (FA) [104] is a swarm-based metaheuristic algorithm inspired by fireflies. Fireflies in nature communicate with and are attracted to the bioluminescent light emitted by other fireflies. The algorithm defines a swarm of fireflies of $size = j$. Each firefly is initiated at random in a location constrained between an upper and lower bound. The algorithm performs a search over t iterations. For each iteration, the position of each firefly is ranked according to a fitness function. The brightness of the firefly is then set according to its fitness meaning that the firefly in the best position is the brightest in the swarm. Next, the brightness of each ith firefly is compared against the brightness of all fireflies within the swarm of size jth. If the ith firefly is less bright than the jth firefly, then the ith firefly moves towards the jth firefly, otherwise, it moves randomly. Distance between the ith and jth firefly is also considered. The further apart two individuals are from each other, the less bright they appear to each other. Once the position of a firefly is updated, its brightness is immediately recalculated based on the fitness of its new position. The downside of FA is that studies have reported that the algorithm can be slow to converge [105] [106] due to the random movements in the search phase.

Genetic Algorithms (GA) [107] is a metaheuristic algorithm that mimics natural selection. The first stage of the algorithm is to generate a population of individuals in random positions. Secondly, over several iterations, until a stop condition is met, each individual's position is evaluated using a fitness function. The individuals are then ranked in order of fitness. If a stop condition is not met, a new population is generated for the next iteration. The new population is generated by adopting three main steps, a section step, a crossover step, and a mutation step. The section step involves selecting the best individuals from the population to become parents. The selection criteria randomly selects individuals. Better performing individuals have a higher probability of selection, simulating survival of the fittest. Next, the crossover step combines two parent individuals to generate offspring which contain features from both parents. Lastly, a mutation step adds randomness to the crossover step by introducing diversity into the population. This is to avoid the population becoming stagnant. Studies have shown that GA algorithms are slower to converge than PSO based algorithms for CNN architecture generation [10].

Ant Colony Optimization (ACO) [108] models the natural foraging nature of ants. As ants find food, they indirectly communicate with other ants by leaving behind a trail of pheromones. Once the food gets found, the ants return following the same pheromone trail, further increasing its scent. The pheromone scent decays over time. Routes containing stronger scents are a good indicator of a close food source. Subsequent ants pick up on the pheromone scent, following it to the food source. This further increases the pheromone levels. The best routes, therefore, have the strongest pheromone levels. ACO models this behaviour by modelling the possible route connections as a graph. The amount of pheromones between two edges on a graph is calculated by Equation 7 where $\Delta\tau_{i,j}^{k}$ is the amount of pheromone for the kth ant, between the graph connection of edge i to j. This is calculated with $\frac{1}{L_k}$ where L_k is the length of the edge between i and j. A calculation based on path length means that shorter paths have stronger pheromones when visited.

$$\Delta\tau_{i,j}^{k} = \frac{1}{L_k} \tag{7}$$

ACO models multiple ants so Equation 8 calculates the pheromones levels left by all ants, at all edges in the graph. The term $(1-\rho)\tau_{i,j}$ simulates evaporation of pheromones, where ρ is a constant.

$$\tau_{i,j}^{k} = (1-\rho)\tau_{i,j} + \sum_{k=1}^{m} \Delta\tau_{i,j}^{k} \tag{8}$$

Cuckoo Search (CS) [109] is a swarm-based metaheuristic algorithm that models the parasitic behaviour of cuckoo birds, which lay eggs in a hosts birds nest. If the host bird discovers the cuckoo eggs, there is a chance that the host will abandon its nest or throw away the cuckoo egg. The authors base CS search behaviour on lévy flights [110]. Lévy flight is a flight mechanism seen in flies whereby travel is performed as a series of straight lines, broken up by sharp 90 turns. Cuckoo search defines three rules 1) A cuckoo will lay one egg at a time in a randomly selected nest. 2) The nest containing the best egg carries over to the next generation 3) The number of available nests is fixed. The probability that a host bird will discover the cuckoo egg is defined with $p_e \in [0,1]$. The algorithms aim is to replace the existing solutions in the nest, i.e the host eggs, with better solutions. The weakness of CS, when compared to PSO, is that CS uses Lévy flight. The movement mechanism of CS is less iterative when compared to PSO [111]. Cuckoo Search has also been shown to easily get stuck in local optimal traps [112]

2.6 Evolving CNN Models with PSO Based Techniques

Evolutionary algorithms such as PSO show superior search capabilities in solving diverse optimization problems. Related studies adapt the PSO algorithm to CNN architecture generation. Evolutionary techniques such as PSO based algorithms for generating CNN architectures is relevant to resource-constrained environments because the search space can be constrained to ensure candidate models will fit within the target environment. Furthermore, the generated model is designed specifically for the task at hand which could lead to more optimal solutions when compared to transfer learning. Before discussing recent deep architecture generation methods, the PSO algorithm is introduced in this section.

In comparison with other optimization methods, PSO shows impressive search capabilities in solving single- and multi-objective optimization problems [113], [114], [115], [116]. PSO when compared to other evolutionary techniques is relatively easy to implement and compute, as indicated by a favourable search time when compared to other techniques outlined in this section. extended Because of this, it has been widely adopted for optimizing CNN models [117] [118].

One of the first studies of applying PSO to CNN generation was IPPSO [13]. The IPPSO model adopts a flexible encoding scheme to address the limitation of the traditional PSO model where particles are required to have a fixed length. The encoding scheme is inspired by IP addressing and subnetting in computer network research. Specifically, an IP address strategy is used to represent the layer parameters as a series of bits. As an example, the kernel sizes within the search range between 1 and 8 are encoded into 3 bits, i.e. a kernel size of 1 is encoded as 001. A subnet is used to identify a layer type, i.e. a convolutional, pooling, or fully connected layer. Evaluated using a variant of the MNIST data set, i.e. MNIST-BI (MNIST with background images), IPPSO has achieved state-of-the-art performance. Experiments were reported as taking on average 2.5 hours to complete for each data set.

A novel PSO variant namely psoCNN [10] was introduced for deep architecture generation. Based on a selection criterion for position updating in a swarm, psoCNN selects the candidate layers from either the global or personal best solution. psoCNN outperforms state-of-the-art models for architecture generation, including IPPSO [13]. It also depicts a low computational cost. One weakness of psoCNN is that the particles are not able to fully explore a search space like they would in the original PSO algorithm. This is because the psoCNN algorithm copies layers from the personal and global best solutions directly. Such a copying strategy significantly reduces search diversity as intermediate positions are not explored. Such a search strategy increases the likelihood of being trapped in local optima.

The sosCNN [14] method was proposed by [14] for deep network generation, which was built upon a previous study of psoCNN [10]. It employed a Symbiotic Organisms Search

(SOS) [119] algorithm instead of PSO for the search of evolving network architectures, by introducing two new strategies. Firstly, a slack gain strategy was proposed for devising architectures with greater depths, in order to overcome a weakness of the SOS algorithm which excessively eliminated deep networks early in the search process. Specifically, the difference between the global best position and the current particle position was calculated, and then a random number was generated for each layer in the network. If the random number was less than 0.5, the difference between the global best and the current position was selected for particle position updating. Otherwise, the original particle position was selected. Secondly, a dissimilar mutation strategy was also introduced which strictly limited the difference in mutations in a way to guarantee that when a block mutation occurred, the resulting block was not too dissimilar. The authors claimed that such a process also helped to ensure faster convergence. The work achieved an error rate of 0.3% on the MNIST data set. However both sosCNN and the previous study of psoCNN [10] do not construct models comprising residual connections, and therefore susceptible to the vanishing gradient problem [3] [38].

PSO has also been applied to the optimization of hand-crafted models such as DenseNet [15]. As an example, Wang et al. [120] proposed a multi-objective PSO method for DenseNet architecture generation, which maximizes accuracy whilst minimizing the computational cost of the devised CNN model, measured in floating-point operations (FLOPS). An encoding scheme was proposed for optimizing the number of dense blocks, layers per block and the growth rate of each block. The architecture search of their model required a high computational cost, i.e. 3 days with the settings of 8 GPUs and a population size of 20, optimized over 20 iterations. The optimized DenseNet model was evaluated on the CIFAR-10 data set and achieved an accuracy rate of 95.51%, which showed an improvement when compared to an accuracy rate of 94.77% for DenseNet-121. However their work did not optimize other key parameters such as kernel size and pooling types within each dense block.

Dutta et al. [121] proposed two PSO variants, namely Qubit Fractional Order PSO (Qubit FO-DPSO) and Qutrit Fractional Order PSO (Qutrit FO-DPSO), with the attempt to identify the best wavelength thresholds to minimise signal noise for Hyperspectral Image (HSI) segmentation. First of all, Improved Subspace Decomposition Algorithm, Principal Component Analysis (PCA), and a Band Selection CNN, were used to conduct discriminative band selection. In addition, their models maintained multiple swarms simultaneously while using quantum parallelism to reduce the computational cost. In the Qubit configuration, each dimension was initialized randomly with a binary value of 0 or 1, whereas in the Qutrit configuration, each dimension was initialized randomly with either 0, 1 or 2. Fractional order (FO) was proposed for velocity calculation which took the last three velocities of each particle into account, and employed a weighted sum of these recent velocities for calculating the new one in the next iteration. Each particle was evaluated using the following three objective functions, i.e. modified Otsu criterion, Masi entropy and Tsallis entropy, for thresholding performance measurement. Moreover, a quantum disaster operation (denoted as D) was proposed in the aforementioned variants to mitigate early stagnation and increase search diversity. This operator deleted particles and even a whole swarm whose fitness scores did not improve over 10 consecutive iterations. It also generated new particles or a new swarm if a particular swarm illustrated enhanced fitness over generations. Evaluated on the Indian Pines, the Pavia University and the Xuzhou HYSPEX data sets, their models achieved measurable improvements in terms of the Peak signal-to-noise ratios and Dice similarity scores in comparison with those of other search methods for HSI segmentation.

Fielding and Zhang [39] proposed a PSO-based method for optimizing skeleton block-based CNN architectures comprising dense connectivity [15]. Their work employed novel weight inheritance learning mechanisms in an attempt to reduce computational costs. Their weight inheritance mechanisms provided continual training to any selection of CNN networks by inheriting the weights for a partially trained CNN model. Specifically, their work initialized a network containing four dense blocks that were subsequently optimized using a modified PSO algorithm with cosine search coefficients. The objective of their PSO model was to discover the optimal number of layers within each dense block, and the growth rate of the overall model. The growth rate controlled the number of filters within the model, i.e the model width. The weight inheritance processes were subsequently applied to any CNN architecture devised by the PSO variant and employed the layer position and the size of its parameter matrix as the search key for weight inheritance. Evaluated on the CIFAR-10 data set, in comparison with related studies such as [122], their method reduced the computational cost of the architecture search from 1000 to 150 GPU hours and also improved the accuracy rate from 89% to 90.28%. But their model did not optimize lower level features such as the kernel size or the number of blocks within the model.

A PSO variant was proposed by Zhang et al. [123] to discover the optimal hyper-parameters when constructing an ensemble hybrid network to perform human action recognition. Each base network within the ensemble model was composed of a GoogLeNet in combination with a bidirectional Long Short-Term Memory (BLSTM) network. Hyper-parameters, such as the learning and dropout rates and the number of hidden units in the BLSTM layers, were optimized using their proposed PSO variant. The PSO operation was guided by hybrid leader signals generated using nonlinear crossover operators, as well as 3D superellipse coefficients, to overcome stagnation. A number of base googLeNet-BLSTM networks were optimized using the PSO method, which was subsequently used to construct ensemble models. Evaluated using several well-known human action data sets (e.g. KTH, UCF50 and UCF101), their ensemble networks showed impressive performance in comparison with those of ensemble models devised by other PSO variants and existing state-of-the-art methods.

Liu et al. [124] developed two PSO-based methods, i.e. PSO-Net and CPSO-Net, for cell-based CNN architecture generation with respect to Hyperspectral Image (HSI) segmentation classification. Their encoding process was used to transform architectures into arrays by embedding information such as connections and operation types between network nodes. In both methods, PSO was used to devise optimal CNN architectures. In particular, in CPSO-Net, a SuperNet was maintained first, which was trained using gradient descent. Each particle subsequently inherited the network weights from those of this fixed SuperNet. In comparison with PSO-Net where each network devised by each particle was trained individually, the SuperNet in CPSO-Net was trained only once per iteration using the gradients of all particles in the swarm to accelerate the optimal network generation process. Evaluated using biased and unbiased HSI data sets, their methods obtained improved accuracy rates as compared with those of existing state-of-the-art studies.

An evolutionary group-based PSO (EGPSO) was proposed by Juang et al. [125] for optimizing weights in Recurrent Neural Networks (RNNs) with respect to the generation of forward walking gait of a hexapod robot. Their model incorporated group-based GA with PSO, which outperformed other GA and PSO methods for optimizing the walking speed of the robot. Tan et al. [126] proposed a PSO variant for optimal hyperparameter selection for a VGG network for melanoma classification. Their PSO variant employed three subswarms guided by distinctive adaptive nonlinear search coefficients, as well as sub-dimension based search for leader enhancement. A wrapper-based feature selection was also conducted using the PSO algorithm for ensemble model construction pertaining to lesion classification. Moreover, a PSO method with elliptical search coefficients was proposed by [127] for hyperparameter fine-tuning of Mask R-CNN for medical image segmentation, while PSO in combination with random walk strategies and FA operations [128] was exploited for K-Means clustering centroid enhancement and deep architecture generation for image segmentation and classification, respectively. PSO-based generative adversarial networks (GANs) were also proposed by [129] for facial image generation. The PSO model was used to optimize the parameters of the generator network to improve training stability. The quality and diversity measurements of generated images were taken into account in the cost function. Evaluated using the CelebA data set, the PSO-enhanced GAN model outperformed the original GAN and other variant methods and overcame the vanishing gradient problem of the original GAN model.

Gao et al. [130] proposed a gradient-priority PSO algorithm for deep network generation for undertaking EEG-based emotion recognition. It addresses the efficiency limitations of automated architecture search in a high-dimensional search space. A hybrid model based on PSO and the gradient descent method is proposed for carrying out a weighted exploration in dimensions of greater importance. The method identifies the optimal settings of both the convolutional and pooling layers. To be specific, the kernel sizes and the number of output channels are optimized for the convolutional layers. For pooling layers, the optimal pooling types and the kernel sizes are identified. Instead of calculating the distance between the current particle and the global best solution, the model computes the maximum gradient position for each particle, which is subsequently used to accelerate the particle movement in the direction illustrating the most impact. The method achieves an impressive performance for deep network generation for emotion recognition in comparison with those from existing studies.

Wang et al. [40] proposed an efficient PSO model, namely EPSOCNN, for deep architecture generation. It employs the classical PSO algorithm to optimize a single network block only, i.e. a dense block, to accelerate the evolutionary process. Proposed in DenseNet [15], a dense block contains 3x3 convolutions and residual connections. Specifically, the EPSOCNN model optimizes the number of output channels within a block using a widening factor, as well as the number of convolutional layers with a search range between 6 and 32 within a single block. Once the best dense block is found, a second-stage process progressively stacks the optimized dense block to identify the optimal number of blocks, in order to construct the final network. With fewer than 4 GPU days, EPSOCNN yields an error rate of 3.58% on the CIFAR-10 data set, with an improvement of 1.12% over that in [131]. Note the study conducted has been based on the optimization of an existing state-of-the-art network block.

2.7 Evolving CNN Models with Other Evolutionary Techniques

Recently, Sun et al. [132] proposed a GA-based deep architecture generation model, namely CNN-GA, to automatically devise networks for image classification. Since the GA-based optimization process commonly employs a fixed-length encoding strategy, where the length of chromosomes is fixed and must be specified beforehand, CNN-GA introduces a variable-length encoding operation to overcome this restriction. The encoding scheme considers both the residual blocks and pooling layers. A residual block contains two convolutional layers with fixed kernel sizes of 3x3 and 1x1, respectively, along with a residual connection. A binary tournament selection mechanism [133] is adopted, whereby two individuals are selected at random based on the fitness scores. A crossover operation is subsequently performed with respect to a random threshold. Mutations are conducted by adding, removing, or modifying layers. Evaluated using CIFAR-10 and CIFAR-100 data sets, CNN-GA outperforms several existing methods such as NASNet [134] and DARTS [135].

Another deep architecture generation model, namely Automatically Evolving CNN (AE-CNN), was proposed in [131]. For deep network generation, the AE-CNN model employs either a ResNet block [3] with three convolutional layers and residual connections, or a DenseNet block [15] with four convolutional layers and residual connections. The pooling layer is designed to perform mean and max-pooling using a 2x2 kernel. The computational cost of the search process is handled by introducing asynchronous computation and caching, which successfully reduces the cost of the fitness evaluation. Based on the CIFAR-10 and CIFAR-100 data sets, the proposed method is able to reduce the computational cost from the 100 GPU days as required by MetaQNN [11] to 35 GPU days. However, their proposed strategies have not explored kernel sizes other than 3x3 in the convolutional layers.

Within the family of neural architecture search (NAS) based algorithms, Kwasigroch et al. [136] aimed to reduce the computational cost when generating CNN model architectures from scratch. A novel network morphism operation is combined with a greedy search algorithm (i.e. hill-climbing) to generate an optimal architecture in malignant melanoma classification. The search mechanism incrementally increases the model size over a number of iterations. Savings in the computational cost are achieved by inheriting the previously trained weights over to the new offspring. The architecture search initially constructs a base model. It consists of a 3x3 convolutional layer, followed by a max-pooling layer, a second 3x3 convolutional layer and finally a Sigmoid activation function. This base model is trained before multiple offspring solutions are created. The offspring networks are created by applying an operation to extend the parent model. The available extension methods include inserting new layers, altering the number of output channels on existing layers, or stacking two layers side by side and applying an addition or concatenation operation to the output of each layer. The offspring models are trained when the training of the parent model is completed. The best offspring model from the current iteration is selected to become the parent for the next iteration. Impressive results are achieved. The entire search process requires 18 GPU hours, as compared with 38 GPU hours required by a related method reported in [131].

A Neural Architecture Search (NAS) method based on Ant Colony Optimization (ACO) [108] called DeepSwarm [137] was applied to the generation of deep convolutional neural networks. One by one, each ant in a colony got placed on the input node. Guided by pheromone signals, the ant selects a node type, followed by node hyperparameters such as the kernel size, filter count and activation layer type. The ant navigates the graph until the maximum model depth is reached. The selected architecture was when evaluated before the ants chosen path pheromone signals got updated. Finally, the overall model with the best accuracy is identified as the optimal model construction. DeepSwarm achieved a mean error rate of 0.46% on the MNIST data set.

In order to better balance the search between exploration and exploitation of the Monarch Butterfly Optimization (MBO) algorithm [138], a new hybrid variant model, namely the MBO-Artificial Bee Colony Firefly Enhanced (MBO-ABCFE) algorithm, was proposed by [139], for deep architecture generation. Their model incorporated MBO with Artificial Bee Colony (ABC) and Firefly Algorithm (FA) to increase search diversity and overcome the premature convergence problems of the original MBO algorithm [140]. Specifically, it diversified global exploration by incorporating the search mechanisms of ABC [141], along with a control parameter which adjusted the intensification. The local exploitation was also increased by adopting the search strategy of FA [142]. In addition, two new parameters were introduced, i.e. an exhaustiveness parameter and a trial parameter. After each iteration, if an individual butterfly solution did not improve, the trial parameter for the butterfly was increased by one. Once the trial parameter exceeded the exhaustiveness parameter, a new individual was randomly initialized. The search exploration was therefore further improved by replacing poor performing individuals stuck in local optima with new solutions. A number of hyperparameters were optimized by their proposed model, i.e. the number of convolutional layers, kernel size, type of activation functions, pooling size, batch size and learning rate. Evaluated using the MNIST data set, MBO-ABCFE devised networks achieved an error rate of 0.34% with an improvement of 0.02% over those yielded by the original MBO algorithm.

Chen et al. [143] proposed a BASCNN method by applying a recently proposed meta-heuristic algorithm, i.e. Beetle Antenna Search (BAS) [144] for the optimization of CNN hyper-parameters. The BAS algorithm models food sensing behaviours of beetles and searches for an optimal solution in the search space using a single search agent. Specifically, the BAS algorithm was used to optimize the initial weights and biases of a LeNet model at the early stage of model training. Their work was evaluated using a brain CT scan data set containing 200 images from subjects in different age groups, half of which included patients with intracranial haemorrhage. The BASCNN model achieved an accuracy rate of 93.93%, outperforming those of existing studies such as [145] and [146]. But their experiments were limited to the optimization of the initial weights and biases of CNNs, without the consideration of optimizing the network structures.

A GeNET model was proposed by [31] for CNN architecture generation based on the GA method. Their work employed an encoding scheme based on a fixed-length binary string. To be specific, this fixed-length binary string encoded the inter-connections between CNN architecture nodes. Each node contained a set of convolutional, batch normalization and Relu layers. Genetic operations such as selection, mutation and crossover mechanisms, were subsequently conducted. In particular, the mutation operation has a low probability of flipping a bit within the fixed-length binary string, thus slightly altering the node connections. The work achieved error rates of 0.34%, 5.39%, and 25.12% for MNIST, CIFAR-10, and CIFAR-100 data sets, respectively.

Architecture generation has also been conducted by combining strategies from multiple evolutionary techniques simultaneously. As an example, Tirumala [147] proposed a multi-population competitive and cooperative neuroevolution method, namely DNN-COCA, for deep neural network architecture generation. Specifically, their model divided a population into two sub-populations P_1 and P_2, and applied a different search strategy in each sub-population. The sub-population P_1 employed a competitive co-evolution search method, whereas P_2 adopted a cooperative search strategy. To maintain search diversity, the work introduced an interpopulation migration strategy that migrated individuals between P_1 and P_2. A table of the best individuals from both populations was maintained and used to generate offspring solutions. The model achieved an accuracy rate of 98.7% on the MNIST data set by evolving the total number of layers within a CNN between 5 and 7 layers. However, such constrained experimental settings compromise their model performance.

Calisto and Lai-Yuen [148] employed a multi-objective evolutionary algorithm based on decomposition (MOEA/D) [149] for the automatic construction of an ensemble of 2D and 3D residual models for medical image segmentation using volumetric medical data. A 2D CNN model extracted in-plane intra-slice information, while a 3D CNN model exploited volumetric inter-slice information. The work employed a multi-objective cost function that minimized the error rate and the number of model parameters simultaneously. The algorithm optimized the number of residual blocks, the number of filters of the first residual block, the kernel size for convolutional layers within each block, the activation function, dropout and learning rates, for both 2D and 3D models. Evaluated using the prostate segmentation task in the PROMISE12 Grand Challenge [150], the model achieved an impressive pixel-wise classification accuracy rate of 89.29%, ranked among the top 10 results for the challenge at the time of publication.

Regardless of the search strategies, the bottleneck in optimizing CNN models is the considerable computational cost in the fitness evaluation of the candidate models. Such a fitness evaluation procedure needs to be repeatedly performed over a significant number of times during the optimization process. Related studies such as [151] indicated that only 1.14% of the candidate models achieve good results, and 88% provide reasonable results, while the remaining illustrate poor performances. The study indicates that the majority of the computational cost has been spent evaluating less optimal networks during the search process. As such, a linear prediction model was proposed in [151] as the performance predictor of deep networks. Such techniques can be combined with any optimal architecture generation process to improve computational efficiency while effectively exploring the search process.

There are also other related studies, such as [152] and [153], which adopt non-evolutionary techniques such as pruning to remove insignificant weights for devising deep networks.

In summary, many approaches to automatic CNN architecture generation exist. Background research indicates that particle swarm optimization (PSO) based approaches to CNN architecture generation converge faster than alternative methods such as Firefly and GA based approaches. Fast convergence is an important consideration with architecture generation as the search space is large owing to a large number of hyperparameter settings, and the search time is slow, due to the need to train CNN models at every particle position to evaluate fitness. Existing state-of-the-art PSO based CNN architecture generation approaches do however contain weaknesses. The weaknesses are 1) the current state-of-the-art PSO based architecture generation technique i.e psoCNN [10] limits the search by copying blocks from solutions in promising positions. Such an approach means that intermediate positions are not explored and the overall approach is more susceptible to the quality of the particle initialisation, 2) psoCNN uses hard-coded rules to ensure the proposed models remain valid. Such rules interrupt the natural particle movement as applying the rule directly modifies the particle position directly, and 3) Existing state-of-the-art PSO architecture generation techniques such as psoCNN [10] and sosCNN [14] do not take advantage of residual connections [3], which means that existing techniques are unable to construct deep CNN models, owing to the vanishing gradient problem [3].

3 Proposed Efficiency Focused CNN Architecture (IoTNet)

3.1 Introduction

The motivation of this research is to design a novel efficiency-focused model specifically for resource-constrained devices which greatly improves accuracy, and reduces computational cost simultaneously. Specifically, a new efficiency focused model is proposed, namely IoT-Net, which improves the trade-off between accuracy and computational cost by avoiding the common pitfall of efficiency-focused related studies which is to perform convolutions as depth-wise separable operations to reduce computation cost. This research reduces the computational cost by factorizing the $3x3$ standard convolutions found in large and highly accurate models into pairs of 1x3 and 3x1 normal convolutions which reduce the number of parameters by 33%. The empirical results indicate that the proposed approach delivers significantly enhanced performance with less computational cost measured as a reduction in FLOPs. The way the proposed model differs from other existing studies is visualized in Figure 11.

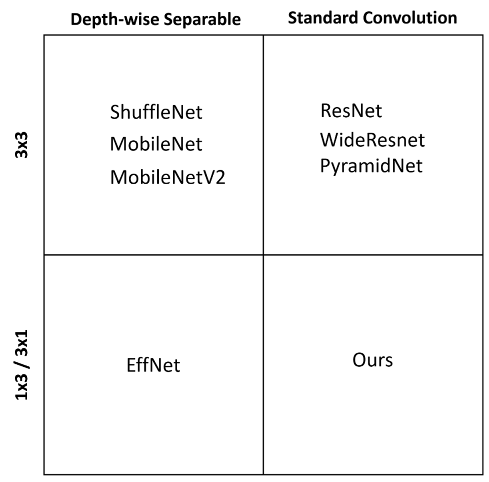


Figure 11: The proposed model dubbed IoTNet is distinctive from other related works as it uses pairs of 1x3 and 3x1 standard convolutions, rather than 3x3 standard convolutions typically found in large models, or depth-wise separable convolutions used in efficiency-focused models.

3.2 Distinction Between Standard Convolutions and Depth-Wise Separable Convolutions

The following section aims to make a clear distinction between a standard convolution found in the proposed model and depth-wise separable convolutions found in related works, pertaining to their differences in methodology and computational cost. Subscripts have been used because the proposed kernels are not square as they are either 1x3 or 3x1 in shape.

3.2.1 Standard Convolution

For an input f of size $Df_1 \times Df_2 \times M$, a standard convolution uses a kernel k which extends the entire depth of the input. The kernel therefore has a size of $Dk_1 \times Dk_2 \times M$. Convolving k with input f produces an output g of size $Dg_1 \times Dg_2$ as seen in Figure 12.

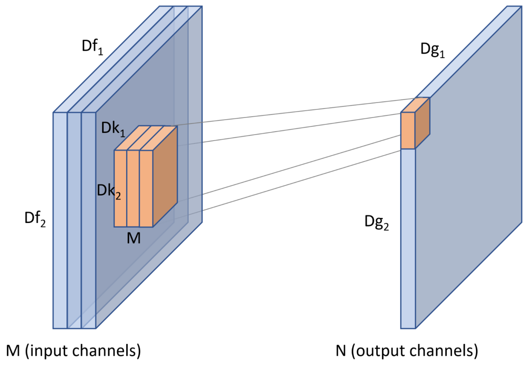


Figure 12: A standard convolution uses a kernel which extends the entire depth of an input.

A total of N such kernels can be used to produce multiple output channels. The computational cost of a standard convolution can be calculated with Equation 9

$$standard = N \cdot Dg_1 \cdot Dg_2 \cdot Dk_1 \cdot Dk_2 \cdot M \tag{9}$$

3.2.2 Depth-Wise Separable Convolution

A depth-wise separable convolution is performed in two stages, i.e., a depth-wise stage, and a point-wise stage. The computational cost of a depth-wise separable convolution is therefore the sum of the computational cost of both stages.

The depth-wise stage uses a kernel k which spans only one channel of input f. M such kernels are used to span the entire depth of the input to produce an intermediate output g of size $Dg_1 \times Dg_2 \times M$ as shown in Figure 13.

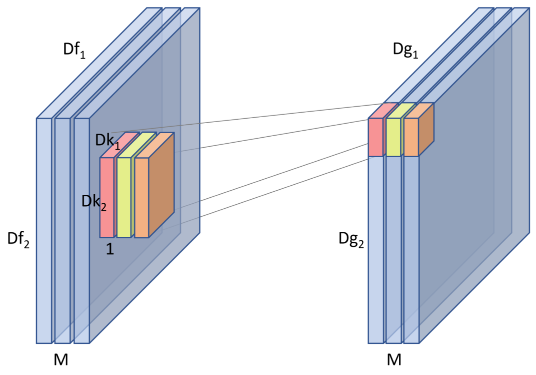


Figure 13: In the depth-wise phase, multiple kernels are used to exploit the entire depth of an input as each kernel only spans one channel.

The computation cost of this phase can be calculated with Equation 10

$$depthwise = M \cdot Dg_1 \cdot Dg_2 \cdot Dk_1 \cdot Dk_2 \tag{10}$$

The point-wise stage combines the intermediary output from the depth-wise stage using a standard convolution, commonly with a 1×1 kernel. As with the standard convolution, you can have N such kernels to produce multiple output channels as shown in Figure 14.

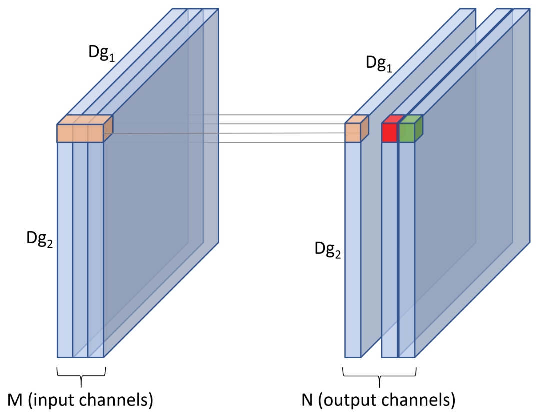


Figure 14: In the point-wise phase, a standard convolution is performed on the intermediate output from the depth-wise phase.

The computational cost of this phase can be calculated with Equation 11

$$pointwise = N \cdot Dg_1 \cdot Dg_2 \cdot M \tag{11}$$

The motivation when using a depth-wise separable convolution is to reduce the computational cost. The cost saving can be calculated as Equation 12 which simplifies to Equation 13.

$$cost = \frac{depthwise + pointwise}{standard} \tag{12}$$

$$cost = \frac{Dk_1 \cdot Dk_2 + N}{N \cdot Dk_1 \cdot Dk_2} \tag{13}$$

It is more likely to perform a convolution as a depth-wise convolution rather than a standard convolution when using a 3x3 kernel than it is when using a 1x3 kernel. This becomes clearer when comparing the savings of both kernel types. Equation 14 shows that a convolution with a 3x3 kernel and 64 channels will require only 12.7% of the total FLOPs a normal convolution would require if performed depth-wise, which is a significant saving. On the other hand, equation 15 illustrates that a convolution with a 1x3 kernel and 64 channels will use 34% of the total FLOPs a normal convolution would require. This means that the cost saving is greater on a 3x3 kernel, therefore the proposed study is more motivated to perform standard convolutions and instead reduce computation through factorisation i.e a 1x3 and 3x1 kernel, replacing a single 3x3 kernel.

$$\frac{Dk_1 \cdot Dk_2 + N}{N \cdot Dk_1 \cdot Dk_2} = \frac{3 \cdot 3 + 64}{64 \cdot 3 \cdot 3} = 0.127 \tag{14}$$

$$\frac{Dk_1 \cdot Dk_2 + N}{N \cdot Dk_1 \cdot Dk_2} = \frac{1 \cdot 3 + 64}{64 \cdot 1 \cdot 3} = 0.34 \tag{15}$$

3.3 The Proposed Efficiency Focused Model

The proposed model consists primarily of groups and blocks. A group is a logical collection of blocks. A group also contains metadata such as to what degree resolution downsampling and widening should be applied. A block is a collection of operations such as convolutions which are performed in a repeatable sequence.

The proposed model has an initial 3x3 convolution, followed by at least one group of blocks. The depth of the proposed model is controlled by increasing or decreasing the number of groups with $g \in \{1,2,3\}$ and controlling the number of n blocks within each group where $n >= 1$. A block consists of batch normalization [154] followed by a pair of 1x3 and 3x1 standard convolutions and contains a residual connection. A block is defined in Equation (16) from [4] as x_l and x_{l+1} represent the input and output of the *l-th* block in the network, respectively. F is a residual function and W_l is the parameter matrix of the block. Each convolution is preceded with a ReLU [155].

$$x_{l+1} = x_l + F(x_l, W_l) \tag{16}$$

The proposed network block is shown in Figure 15.

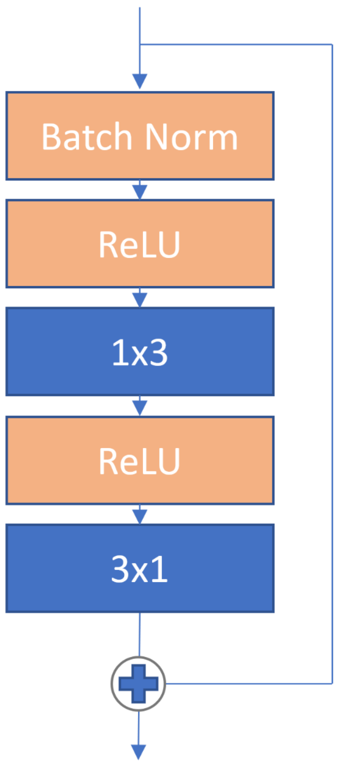


Figure 15: The proposed network block contains a batch normalization, followed by a pair of 1x3 and 3x1 standard convolutions. Each convolution is preceded with a ReLU. Each block also contains a residual connection [3].

The width within the proposed model is controlled with a widening factor of k. The first block of each group is responsible for increasing width. The initial 3x3 convolution has a width of $floor(16*k)$. Group one has a width of $floor(16*k)$, while group two has a width of $floor(32*k)$, and group three has a width of $floor(64*k)$. Except for the initial 3x3 convolution, these settings were taken from related studies of [4].

Resolution downsampling is performed in groups two and three if they are present. The first blocks of groups two and three reduce the resolution by performing average pooling using a 2x2 filter, which halves the output resolution. Figure 16 shows how depth, width and resolution can be controlled, and how the proposed model is made up of g groups, containing n blocks.

The linear layer which performs final classification is preceded by batch normalization, ReLU and average pooling.

So that the benefits of the proposed model could be isolated, the experiments in this research did not use the data augmentation policies learnt through related works, such as AutoAugment [86]. Instead, data augmentation was performed by using the mean/std normalization of all data sets so that the architectures themselves can be compared fairly. For the CIFAR-10 data set, the data augmentations from [4] are adopted for fair comparison i.e horizontal flip, random crop and padded by 4 pixels. Missing pixels added through padding are repopulated using reflections of the original input image.

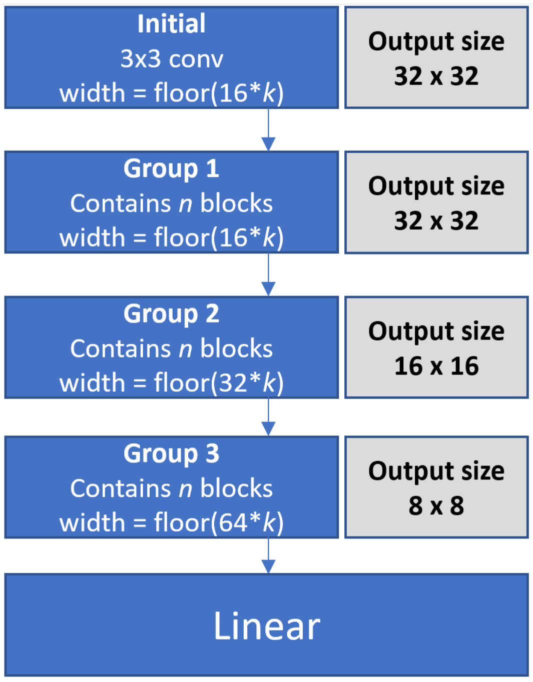


Figure 16: The network width is controlled by a widening factor k. Resolution is reduced within the first blocks of groups two and three if present.

3.3.1 Approach to Identify Candidate Models

In this research, multiple filtering steps are used to reduce the architecture search space and identify the optimal network configurations for each test data set. The detailed process is provided below.

Step 1 – Calculate the FLOPs for all combinations of groups, i.e., $g \in \{1,2,3\}$, number of blocks per group, i.e., $n \in \{1,2,3,4,5\}$, data set classes, i.e., $c \in \{10,43\}$, and the widening factor in the range of $[0.1,2.0]$ in intervals of 0.01.

Step 2 – Filter the results down to networks within a target FLOP range. The range is set to between 50% and 100% of the FLOPs of the smallest baseline benchmark model for each data set. For example, the ShuffleNet large model for CIFAR-10 has the smallest number of FLOPs, i.e., 11.1 million (see Table 2). Therefore, the filtering range for candidate model selection would be between 5.5 and 11.1 million FLOPs.

Table 2: Evaluation results for the CIFAR-10 data set grouped by network sizes in FLOPs. The first group contains larger configurations, while the second group comprises smaller ones.

Model	Widening Factor k	Mean Acc	Mil. FLOPs
EffNet V1 large	0.99	85.02%	79.8
MobileNet large	0.14	78.18%	11.6
ShuffleNet large	0.14	77.90%	11.1
EffNet V1	0.14	80.20%	11.4
EffNet V2	0.22	81.67%	18.1
MobileNetV2	0.20	76.47%	16.4
IoTNet-3-4	**0.7**	**89.9%**	**9.9**
MobileNet	0.07	77.48%	5.8
ShuffleNet	0.06	77.3%	4.7
IoTNet-3-2	**0.68**	**87.19%**	**4.2**

Step 3 – Narrow the results down further by selecting the minimum and maximum widths for every unique combination of g and n. This process results in a list of configurations that contain two candidate models for each unique combination of g and n.

Step 4 – Train the narrowed list of model configurations obtained from Step 3 using a reduced epoch count of 25 to reduce training cost. Then the trained models are tested with the test data set.

Step 5 – Finally select the most promising models (e.g., 2–3 models) based on the test accuracy rates obtained in Step 4 as candidate models for full training with 200 epochs.

Automatic architecture generation techniques, such as Particle Swarm Optimization-based deep CNN model generation, will also be explored in future directions.

3.3.2 Complexity Analysis

The main indicator of computational cost, used in efficiency-focused related studies, such as the benchmark models MobileNet [5], MobileNetV2 [7], ShuffleNet [6] and EffNet [9], is to report the number of floating-point operations, i.e., FLOPs. Therefore, the same indicator is adopted for direct computational cost comparison.

Influential studies such as [5] highlighted that computational cost depends multiplicatively and therefore varies based on the number of FLOPs. The number of FLOPs for a standard convolution as used in this study depends on the number of input channels M, the number of output channels N, the kernel size $Dk_1 \cdot Dk_2$ and the feature map size $Df_1 \cdot Df_2$, as shown in Figure 12. Full details of how convolutional cost is calculated in terms of FLOPs, and a complexity analysis of the proposed modifications and customizations, i.e., cost differences between standard and depth-wise convolutions, and differences between 3x3 and 1x3 convolutions, are provided in Section 3.2. The impact of selecting different widening factors W to the computational cost of a convolution within the proposed model can be compared by scaling Equation 9 from [5] with W, as illustrated in Equation 17. This shows that for the proposed models which share the same numbers of groups and blocks, the computational cost of convolutions scales proportionately with W. In other words, when other network configurations remain intact, the computational cost increases as the widening factor scales up and vice versa.

$$cost = N \cdot Dg_1 \cdot Dg_2 \cdot Dk_1 \cdot Dk_2 \cdot M \cdot W \tag{17}$$

3.4 Experimental Studies

The experimental studies are presented in this section. In Section 3.4.1 a detailed overview of the test data sets employed in the experiments, as well as the model training scheme is provided. Section 3.4.3 evaluates the proposed model against efficiency focused related studies. Section 3.4.7 evaluates the proposed model against models containing 3x3 convolutions. Section 3.4.11 evaluates the trade between accuracy and computational cost for the proposed 1x3 and 3x1 standard convolution approach, with that of a more traditional 3x3 standard convolution. An in-depth model and result analysis in also provided in Section 3.4.12.

3.4.1 Data Sets

The data sets in the experiments include CIFAR-10 [18], SVHN [44] and GTSRB [45]. These data sets offer realistic and varied representations of the types of image classification problems that could be encountered in resource-constrained environments. All these employed data sets have pre-defined training and test sets meaning that all benchmark models and the proposed model have been trained and tested under the same experimental settings, i.e., using the same data splits, samples and image resolutions. The selected data sets are summarized in Table 3 and introduced in more detail in the following subsections.

The CIFAR-10 data set [18] consists of 60,000 images in 32x32 resolutions. They are split into 50,000 and 10,000 samples for training and test, respectively. Each image is categorized as one of the ten classes, including airplane, automobile, bird, cat, deer, dog, frog,

Table 3: Overview of the data sets used in the experiments.

Data Set	Total Sample Size	Training Samples	Test Samples	Image Resolution
CIFAR-10	60,000	50,000	10,000	32x32
SVHN	99,289	73,257	26,032	32x32
GTSRB	51,839	39,209	12,630	32x32

horse, ship and truck. Figure 17 illustrates some example images extracted from this data set.

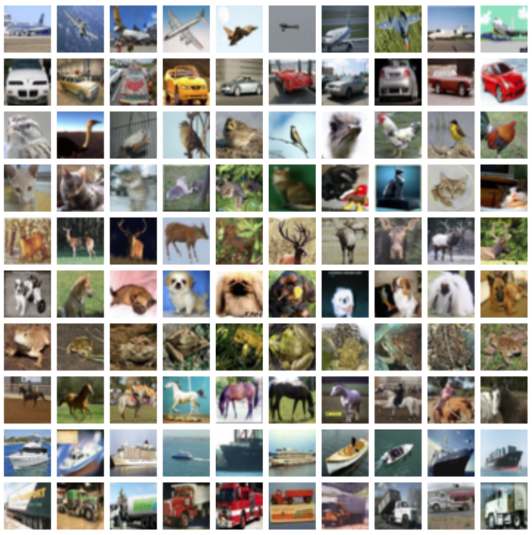


Figure 17: Example images extracted from the CIFAR-10 data set.

The SVHN data set [44] contains house numbers obtained from Google Street View images. It is divided into 73,257 training and 26,032 test images. A 32x32 crop of the original images is used, which produces a MNIST-like data set. In this format, each image contains one digit of interest belonging to one of 10 classes, i.e., a number between 0–9, along with some distracting digits in both sides of each image. Some example images in the SVHN data set are shown in Figure 18.

The GTSRB [45] data set is composed of 51,839 images of traffic signs covering 43 different classes of signs including stop, no entry and speed limits. Images within the same class are of different physical signs with various lighting conditions and image qualities. The data set is split into 39,209 and 12,630 samples for training and test, respectively. Figure 19 illustrates some example images from this data set.

Moreover, GTSRB illustrates imbalanced class distributions with relatively small numbers of examples for some of the classes as shown in Figure 20. Such characteristics make the data set challenging as it is prone to overfitting.

3.4.2 Training Scheme

When training using the CIFAR-10 [18] and GTSRB [45] data sets, stochastic gradient descent (SGD) is used as the optimizer and cross-entropy loss is used for the loss function.

Figure 18: Example images extracted from the SVHN data set.

Figure 19: Example images extracted from the GTSRB data set.

A total of 200 epochs are used for model training. The initial learning rate lr is set to 0.1, and dropped by $lr * 0.2$ at epochs 60, 120 and 160. For the SVHN [44] data set, the Adam optimizer [156] is used, along with cross-entropy loss. A total of 200 epochs with a fixed learning rate of 0.001 have been applied. The above experimental settings are obtained by trial and error to achieve the best model performance. The epoch steps of 60, 120 and 160 were selected based on related works using the same data sets [4]. The learning rate and the degree to which the learning rate was reduced were selected by comparing various settings with a grid search. The mean accuracy over a total of 5 runs is reported and used as the main criterion for comparison. The proposed model has been implemented using PyTorch [157].

3.4.3 Comparison Against Efficiency-Focused Benchmark Models

A comprehensive evaluation has been conducted to compare the proposed block architectures against the baseline efficiency-focused models, i.e., EffNet [9], MobileNet [5], MobileNetV2 [7] and ShuffleNet [6]. The evaluation also compares the proposed approach directly against

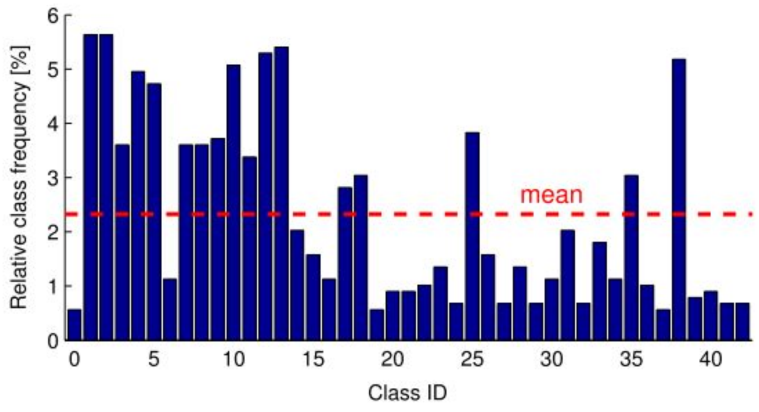


Figure 20: The imbalanced class distributions within the GTSRB data set.

a standard 3x3 convolution method typically used in state-of-the-art accuracy-focused models. The depth settings for the proposed model is denoted as IoTNet-*g*-*n* where *g* is the number of groups, and *n* is the number of blocks within each group. IoTNet-3-2 for example contains three groups, each of which contains two blocks. The same experimental settings are used to ensure a fair comparison, i.e., by using the aforementioned data sets with predefined training and test sets and input resolutions. The experimental results are presented separately for each test data set.

The evaluation results of the baseline networks were obtained from the work of EffNet [9]. The authors of EffNet constructed models of comparable sizes by adjusting the width of the network using a widening factor, adding additional layers or a combination of both. A summary of the models with brief configuration descriptions employed for performance comparison in this research is provided in Table 4. Further details can be obtained from their original studies [9].

3.4.4 Evaluation Using CIFAR-10

The baseline models are split into two categories according to model sizes, measured in FLOPs. The multi-filtering steps as discussed in Section 3.3.1 is adopted to identify suitable candidate models for each network category, which contain fewer FLOPs than those of the baselines. Three candidate models of the proposed approach were identified and trained with 200 epochs on CIFAR-10 for both large and small network configurations, respectively. The detailed results are shown in Table 5.

As illustrated in Table 5, all top candidate models of the proposed approach contain three groups since those with two groups performed worse than such networks, while those with one group performed worse than the models with two groups. This indicates that multiple downsampling stages and network depth are important factors. Candidate models favoured a balance between depth and the widening factor owing to the comparatively challenging nature of CIFAR-10. The empirical results also indicate that shallow but wide models, or deep and narrow models performed worse.

Table 2 shows the detailed results of the best candidate models and the baseline networks

Table 4: A summary of models used for evaluation. Two variations of EffNet introduced by [9] as EffNet V1 and EffNet V2 are included.

Model Name	**Brief Description**
IoTNet-*g*-*n*	The proposed model with *g* as the number of groups, and *n* as the number of blocks per group
EffNet V1	An implementation of EffNet [9]. Model architecture contains 1x3 and 3x1 depth-wise separable convolution and pooling-based blocks
EffNet V1 large	As per EffNet V1 with two additional layers and more channels
EffNet V2	As per EffNet V1, introduced also in [9] in response to MobileNetV2, model contains minor changes relating to network expansion, extension rates (depth and width) and the replacement of ReLU on the point-wise layers with leaky ReLU
MobileNet	An implementation of MobileNet [5] of varying widths. Model architecture contains 3x3 depth-wise separable convolutions
MobileNet large	As per MobileNet implementation with two extra layers
MobileNetV2	An implementation of MobileNetV2 [7] of varying widths. Model contains 3x3 depth-wise convolutions and inverted residual structures where shortcut connections are between bottleneck layers
ShuffleNet	An implementation of ShuffleNet [6] of varying widths. Model contains 3x3 depth-wise convolutions in addition to point-wise group convolution and channel shuffle
ShuffleNet large	As per ShuffleNet implementation with two extra layers

for the CIFAR-10 data set while Table 6 indicates the performance improvements of the best performing candidate models over the benchmark networks. The results in both tables are split into two categories according to model sizes.

Within the first groups (i.e., the larger networks) of Tables 2 and 6, a comparison is provided that compares the proposed best candidate model against larger versions of the efficiency-focused benchmark networks. The best performing benchmark baseline model is EffNet V1 large with an accuracy rate of 85.02%, with 79.8 million FLOPs. The proposed model achieves an accuracy rate of 89.9% with 9.9 million FLOPs. It outperforms EffNet V1 large by 4.88% in terms of accuracy with 87.59% fewer FLOPs. The proposed model also delivers a considerable improvement in terms of accuracy compared to MobileNetV2 with a 13.43% accuracy improvement, with 39.63% fewer FLOPs.

Table 5: Accuracy of the best candidate models found using multi-filtering search, then trained and tested on CIFAR-10. The first group contains networks with larger configurations, while the second group comprises smaller ones.

Model	Widening Factor k	Mean Acc	Mil. FLOPs
IoTNet-3-2	1.08	89.79%	11
IoTNet-3-4	**0.7**	**89.9%**	**9.9**
IoTNet-3-3	0.66	88.98%	6.2
IoTNet-3-2	**0.68**	**87.19%**	**4.2**
IoTNet-3-2	0.5	81.47%	2.6
IoTNet-3-3	0.41	83.49%	2.5

Table 6: The improvements of the proposed best model for the CIFAR-10 data set over the state-of-the-art networks, grouped by network sizes.

Model	Acc Improvement	FLOPs Saving
EffNet V1 large	4.88%	**87.59%**
MobileNet large	11.72%	14.66%
ShuffleNet large	12.0%	10.81%
EffNet V1	9.7%	13.16%
EffNet V2	8.23%	45.3%
MobileNetV2	**13.43%**	39.63%
MobileNet	9.71%	**27.59%**
ShuffleNet	**9.89%**	10.64%

Within the second group (i.e., the networks with smaller configurations), a comparison between the proposed model and smaller versions of the efficiency-focused benchmark models is provided. The best performing baseline model is MobileNet with an accuracy rate of 77.48% and 5.8 million FLOPs. The proposed model achieves an accuracy rate of 87.19% with 4.2 million FLOPs, i.e., an improvement of 9.71%, with 27.59% fewer FLOPs against those of MobileNet. The proposed model also beats ShuffleNet with a 9.89% accuracy improvement, with 10.64% fewer FLOPs.

The results also indicate that the difference between MobileNet and MobileNet large, and the difference between ShuffleNet and ShuffleNet large, in terms of performance when scaling up from smaller to larger sizes, result in less than a 1% improvement in accuracy, respectively. On the contrary, the performance improvement of the proposed models, i.e., between IoTNet-3-4 and IoTNet-3-2, is by 2.71% indicating that diminishing returns when scaling up MobileNet and ShuffleNet, which the proposed model overcomes.

3.4.5 Evaluation Using SVHN

A multi-filtering search strategy is adopted to identify suitable candidate models which contain fewer FLOPs than those of the baselines. Two top candidate models were subsequently identified and trained with 200 epochs on SVHN. The evaluation results on the test set are shown in Table 7.

Table 7: Accuracy of the best candidate models found using the multi-filtering search, then trained and tested on SVHN.

Model	Widening Factor k	Mean Acc	kFLOPs
IoTNet-3-5	**0.14**	**89.22%**	**499.7**
IoTNet-3-2	0.21	88.4%	474.3

As indicated in Table 7, the candidate models containing three groups performed the best which again indicates that multiple downsampling stages and depth are influential factors. Owing to the less challenging nature of SVHN, the candidate models favoured depth over width. Since SVHN contains digits which vary much less between samples than a more general data set would such as CIFAR-10, this indicates that fewer filters are required to extract fine detail and perform classification well. Therefore, less width was required.

Tables 8 and 9 compare the best proposed candidate model against efficiency-focused benchmark models on the SVHN data set. Motivated by the related research [9] where the comparison was conducted using one category of smaller networks owing to the simplicity of the problem, this research makes a comparison with SVHN using a similar style, i.e., purely with the smaller network category. As illustrated in Tables 8 and 9, the best performing benchmark baseline model is EffNet V1 with an accuracy rate of 88.51%, with 517.6 kFLOPs. The proposed model achieves an accuracy rate of 89.22%, with 499.7 kFLOPs, and outperforms EffNet V1 by 0.71% in terms of accuracy with 3.46% fewer FLOPs. It also makes a considerable improvement, i.e., 6.49%, in terms of accuracy with 31.84% fewer FLOPs as compared with those of ShuffleNet. The proposed model is also able to make a significant reduction, i.e., 57.03% in FLOPs, when compared against MobileNetV2 while also improving accuracy by 2.51%. The empirical results indicate that the proposed architecture substantially reduces the trading of accuracy over computational cost by making significant reductions in FLOPs while improving performance across the board.

3.4.6 Evaluation Using GTSRB

The baseline models are split into two categories according to model sizes, measured in FLOPs. A multi-filtering search method was adopted to identify suitable candidate models with fewer FLOPs than those of the baselines, for each network category. Three candidate models were identified for each network configuration, which was subsequently trained with 200 epochs and tested on GTSRB. The detailed results are shown in Table 10.

Table 8: Evaluation results for the SVHN data set.

Model	Widening Factor k	Mean Acc	kFLOPs
EffNet V2	0.34	87.3%	1,204.2
MobileNetV2	0.33	86.71%	1,162.8
EffNet V1	0.14	88.51%	517.6
MobileNet	0.22	85.64%	773.4
ShuffleNet	0.21	82.73%	733.1
IoTNet-3-5	**0.14**	**89.22%**	**499.7**

Table 9: The improvements of the proposed best model for the SVHN data set over the state-of-the-art networks.

Model	Acc Improvement	FLOPs Saving
EffNet V2	1.92%	**58.5%**
MobileNetV2	2.51%	57.03%
EffNet V1	0.71%	3.46%
MobileNet	3.58%	35.39%
ShuffleNet	**6.49%**	31.84%

As shown in Table 10, all top candidate models contained three groups. They outperformed all the networks with one group or two groups, which ascertains the importance of multiple downsampling stages and network depth. The candidate models required less width than on those used for CIFAR-10, yet more width than those tested upon SVHN. Since GTSRB is imbalanced with comparatively more classes (i.e., 43) and contains images with a range of lighting conditions, it is more challenging than SVHN. Therefore, more filters are required in the models than those used in SVHN. On the other hand, GTSRB consists of images with road traffic signs which are comparatively consistent in design and have fewer variations. Thus, it is less challenging than CIFAR-10. Therefore, fewer filters are required than those used in CIFAR-10. Also, the results indicate that shallow but wide, or deep and narrow models performed worse. As an example, the empirical results in Table 10 indicate that when constructing small models, depth must be compromised with width. This can be observed by the improvement in performance when reducing the number of blocks per group from three to two while increasing width.

Tables 11 and 12 compare the best candidate model for each network category against efficiency-focused benchmark models on the imbalanced GTSRB data set. For the larger network comparison, the best performing benchmark baseline model is MobileNetV2 with an accuracy rate of 90.74%, with 710.7 kFLOPs. The proposed model achieves an accuracy rate of 93.17%, with 531.0 kFLOPs. It outperforms MobileNetV2 by 2.43% in terms of accuracy with 25.28% fewer FLOPs. It also makes a significant improvement, i.e., 5.02%, in terms of accuracy with 0.38% fewer FLOPs as compared with those of MobileNet. The

Table 10: Accuracy of the best candidate models found using the multi-filtering search, then trained and tested on GTSRB. The first group contains networks with larger configurations, while the second group comprises smaller ones.

Model	Widening Factor k	Mean Acc	kFLOPs
IoTNet-3-2	**0.22**	**93.17%**	**531.0**
IoTNet-3-5	0.15	90.57%	531.5
IoTNet-3-3	0.18	91.84%	427.1
IoTNet-3-3	0.15	88.25%	342.1
IoTNet-3-3	0.13	88.72%	323.9
IoTNet-3-1	0.24	73.33%	310.3
IoTNet-3-2	**0.18**	**88.82%**	**301.6**

proposed model is also able to make a significant reduction in FLOPs, i.e., 24.63%, when compared against EffNet V2 while also improving accuracy by 2.77%. The empirical results also indicate that when scaling down to 301.6 kFLOPS, it was not possible to increase the width beyond 0.18 but at the same time also staying within the target FLOP range of below 344.1 kFLOPs. Therefore, further studies will be conducted around different widening schemes to address this.

Table 11: Evaluation results of the GTSRB data set. The results are grouped by network sizes in FLOPs. The first group contains larger networks, with the second group showing comparatively smaller models.

Model	Widening Factor k	Mean Acc	kFLOPs
EffNet V2	0.3	90.4%	704.5
MobileNetV2	0.31	90.74%	710.7
MobileNet	0.23	88.15%	533.0
ShuffleNet	0.23	88.99%	540.7
IoTNet-3-2	**0.22**	**93.17%**	**531.0**
EffNet V1	0.15	**91.79%**	344.1
IoTNet-3-2	0.18	88.82%	301.6

3.4.7 Evaluation Against 3x3 Standard Convolutions

To prove the effectiveness of the proposed architecture against a scaled-down state-of-the-art model based on the popular 3x3 standard convolution, a 3x3-based model is constructed by replacing the proposed 1x3 and 3x1 pairs with a 3x3 convolution for comparison. The proposed model and its 3x3 standard convolution counterpart are then scaled down to contain between 1 to 10 million FLOPs. The 3x3 configuration closely resembles scaled-down variants of popular models proposed by [3, 4]. With some minor alterations to the width

Table 12: The improvements of the proposed best model for the GTSRB data set over the state-of-the-art networks, grouped by network sizes.

Model	Acc Improvement	FLOPs Saving
EffNet V2	2.77%	24.63%
MobileNetV2	2.43%	**25.28%**
MobileNet	**5.02%**	0.38%
ShuffleNet	4.18%	1.79%
EffNet	−2.97%	12.35%

calculation, it would also resemble the architecture proposed by [56]. Both models are then trained on the data sets CIFAR-10, SVHN and GTSRB as discussed earlier.

3.4.8 Evaluation Against 3x3 Standard Convolution-based Models Using CIFAR-10

Figure 21 demonstrates that on the CIFAR-10 data set, the proposed model offers significantly improved accuracy rates over its 3x3 counterpart in all cases when scaled between 1 and 10 million FLOPs. The empirical results also indicate that scaling both model variants to sizes greater than 3 million FLOPs results in an improvement of accuracy, but with greater diminishing returns between the model complexity and the observed accuracy improvement.

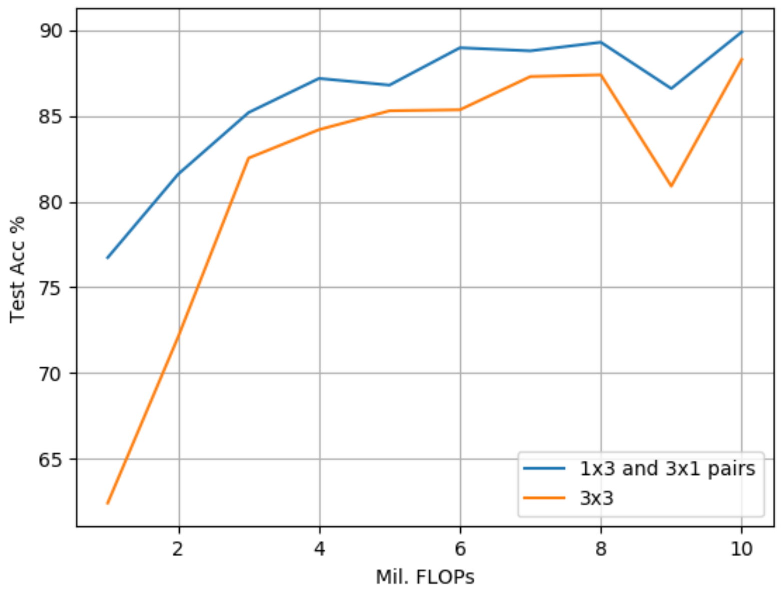


Figure 21: CIFAR-10: The proposed model based on 1x3 and 3x1 convolution pairs compared with a 3x3-based approach. Both variants are scaled to match in terms of FLOPs ranging from 1 to 10 million.

3.4.9 Evaluation Against 3x3 Standard Convolution-based Models Using SVHN

Figure 22 compares the proposed model against its 3x3 counterpart on the SVHN dataset. The empirical results indicate that due to SVHN representing a simpler problem when compared to CIFAR-10, scaling the 3x3 model to more than 3 million FLOPs does not result in any significant performance improvements. Also, the accuracy rate achieved by the 3x3 model when scaled to 3 million FLOPs is surpassed by that of the proposed model containing just 1 million FLOPs. This indicates a significant reduction in computational cost by using the proposed model. For experiments scaled above 6 million FLOPs, the proposed model achieves greater accuracy when using just 5 million FLOPs, which represents another significant reduction in computational cost.

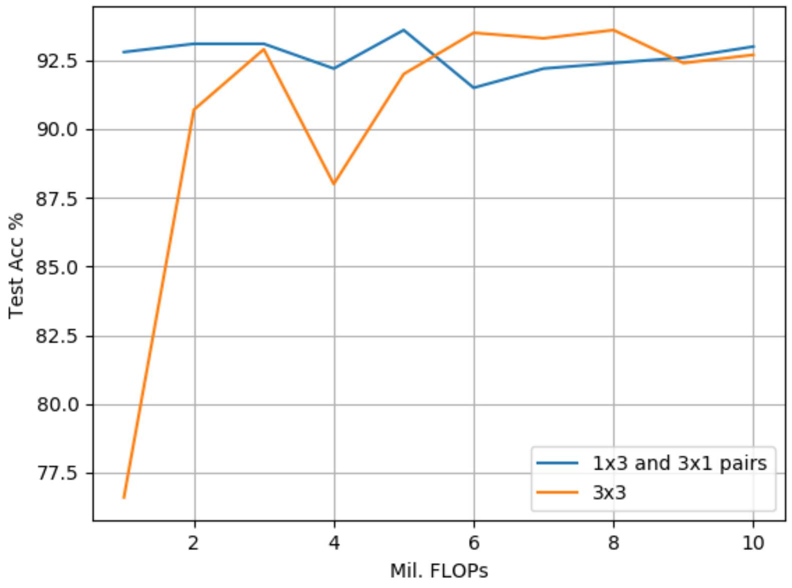


Figure 22: SVHN: The proposed model based on 1x3 and 3x1 convolution pairs compared with a 3x3-based approach. Both variants are scaled to match in terms of FLOPs ranging from 1 to 10 million.

3.4.10 Evaluation Against 3x3 Standard Convolution-Based Models Using GTSRB

Figure 23 compares the proposed model against its 3x3 counterpart on the GTSRB data set. The empirical results indicate that scaling either model larger than 3 million FLOPs does not result in any significant real-world accuracy gains. The empirical results for models containing between 1 and 3 million FLOPs indicate significant accuracy improvements with the proposed model outperforming its 3x3 counterpart throughout. While both models with 4 million FLOPs result in the same accuracy rates, the proposed model scaled above 4 million FLOPs again shows superior performance over its 3x3 counterpart.

3.4.11 Computational Cost Comparison

A model's suitability for deployment on a resource-constrained environment is evaluated by measuring the time and memory utilisation required to process one image from a batch size of 128 images with 32x32 resolutions. The tests are performed using the proposed best performing models obtained from Tables 2, 8 and 11. The time and memory utilisation is measured using a Raspberry Pi 3 Model B+ device and compare them against those of a desktop

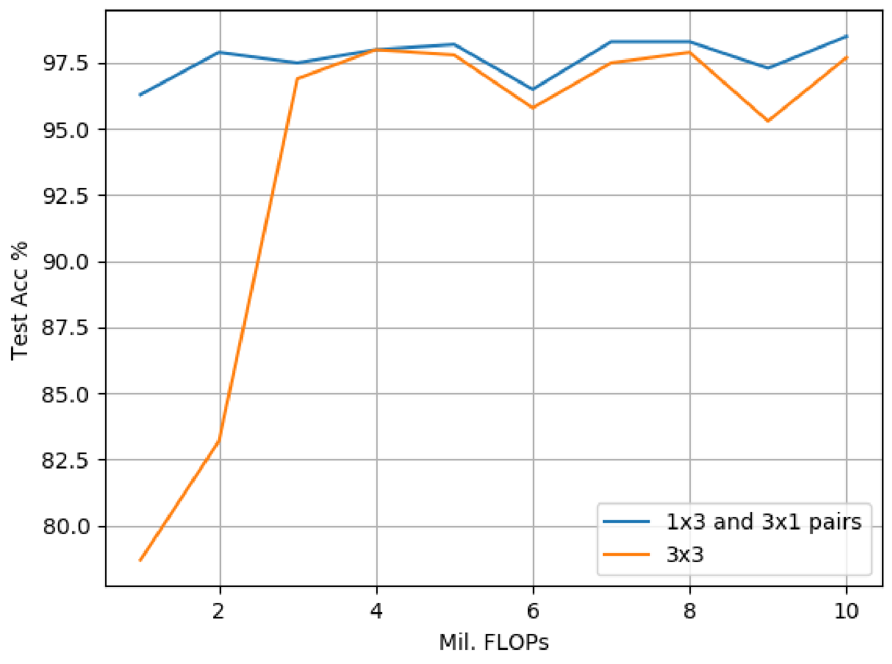


Figure 23: GTSRB: The proposed model based on 1x3 and 3x1 convolution pairs compared with a 3x3-based approach. Both variants are scaled to match in terms of FLOPs ranging from 1 to 10 million.

PC. The specifications of both devices can be found in Table 13, while the results of the tests are recorded in Table 14. All tests are performed using the CPUs, as resource-constrained environments often lack GPUs. The processing time per image for both environments is calculated by processing a batch of 128 images. The time is then divided by 128 to give a mean elapsed per image.

Table 13: Specifications and environmental settings of the desktop PC and Raspberry Pi.

Device	**CPU**	**Memory**	**Operating System**	**Library**
PC	I7-2600k @ 4GHz	16GB	Ubuntu 18.04 LTS	PyTorch 1
Raspberry Pi 3	ARM Cortex @ 1.4GHz	1GB	Raspbian Buster 4.19	PyTorch 1

As expected, the processing time of the Raspberry Pi is longer than that of the desktop PC owing to the significantly faster CPU in the PC. However, the empirical results indicate that the time required to process one image on the Raspberry Pi is very reasonable, ranging between 4.06 ms on the smallest proposed model and 87.5 ms on the largest proposed model. The empirical results also indicate that indeed time is correlated with FLOPs as the slowest proposed model was also the largest in FLOPs. The space requirements across all data sets were well within the bounds of both devices meaning that multiple models could comfortably be deployed within both environments simultaneously. Space could be further reduced if required by decreasing the batch size from 128 to a suitable lower value.

Table 14: Comparison of time and space required to process one image from a batch of 128 between a PC and Raspberry Pi. Time is reported as the time taken to process one image, in milliseconds.

Model	Widening Factor	Data Set	kFLOPs	Memory (MB)	Pi - time (ms)	PC - time (ms)
IoTNet-3-4	0.7	CIFAR-10	9900	392	87.50	0.78
IoTNet-3-2	0.68	CIFAR-10	4200	192	46.09	0.39
IoTNet-3-5	0.14	SVHN	499.7	15	5.94	0.20
IoTNet-3-2	0.22	GTSRB	531.0	27	4.61	0.13
IoTNet-3-2	0.18	GTSRB	301.6	14	4.06	0.16

3.4.12 Discussion

The related studies such as [5, 6, 7] employ depth-wise separable convolutions as a strategy to reduce computational cost, which has proven to be successful pertaining to cost when compared with much larger, state-of-the-art standard convolution-based models such as ResNet. However, in comparison with the most recent works such as [9, 8], they are still quite large, e.g., the MobileNet models contain between 41 and 569 million FLOPs. For much smaller models, more suited to constrained IoT devices, the proposed approach of trading accuracy with computational cost by factorizing 3x3 standard convolutions into pairs of 1x3 and 3x1 standard convolutions leads to significant improvements over the efficiency-focused benchmark models. The empirical results indicate that depth-wise separable convolution-based networks scale down worse than the proposed approach due to a lack of parameters at smaller scales, as demonstrated in the experimental studies. This detrimentally impacts their model training processes as well as their performance as indicated within Tables 2, 8 and 11.

Comparing the proposed approach against scaled-down 3x3 standard convolution-based models as illustrated in Figures 21–23, the empirical results indicate that on all data sets, the proposed model outperforms its 3x3 counterpart greater at smaller scales, i.e., with less than 3 million FLOPs. One explanation for this was provided by [3], which highlighted that the important factors to overall model accuracy are network depth and multiple downsampling stages. In other words, deep models that perform downsampling in multiple stages are likely to lead to promising accuracy rates, while shallower models with fewer downsampling operations are more inclined to suffer from poor performance. This is confirmed by the findings in Tables 5, 7 and 10 where the best proposed candidate models always contained three groups. A second explanation pertaining to an interesting side effect to the proposed approach of factorizing a 3x3 standard convolution into a pair of 1x3 and 3x1 convolutions is that it also doubles model depth. The empirical results indicate that this increase in depth was a key factor leading to the significant performance improvements observed in this study. An increase in depth can, however, lead to overfitting within larger models, i.e., over 3 million FLOPs, which is indicated by a narrower improvement in performance at larger scales against its 3x3 counterparts. As the proposed model is designed to improve performance at smaller scales for resource-constrained environments, this trade-off is believed to be acceptable but could be addressed in future directions by incorporating cutting-edge data augmentation strategies such as [86].

4 Proposed PSO Based Architecture Generation for Image Classification

4.1 Introduction

The motivation of this research is to design a automatic procedure for deep CNN model generation. The following novel contributions are proposed. 1) An encoding scheme which ensures particle positions form architecturally valid and solutions is proposed. Such a scheme avoids the need for additional hard coded rules found in related studies [10], and reduces wasteful function evaluations. 2) A particle velocity updating scheme is proposed to effectively guide each particle through a complex search space so that more accurate networks can be discovered quicker. Success is evaluated against the current state-of-the-art algorithms [10] [11] and 3) Propose how to design efficient CNN models using an algorithm which is both easy to understand and fast to run, so that the approach can be easily exploited in both academia and industry settings with limited specialised knowledge, while not compromising the overall performance. Fig. 24 illustrates an overview of the proposed deep architecture generation model.

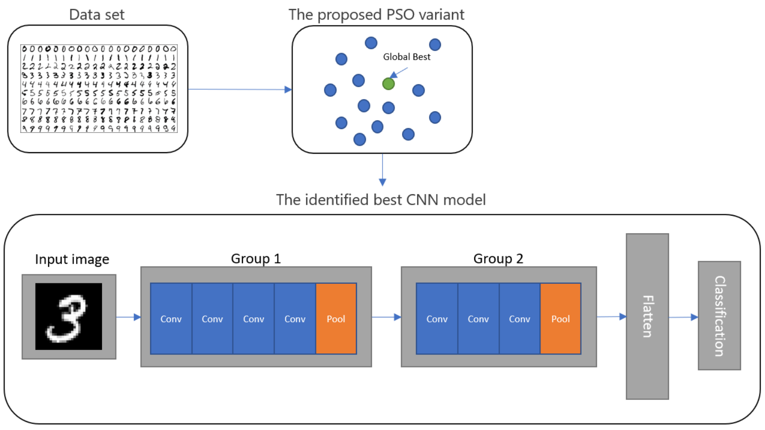


Figure 24: The proposed system architecture where the identified best model is indicated by the global best solution

4.2 The Proposed Approach for Deep Architecture Generation

The proposed PSO variant incorporates a group-based encoding strategy as well as search operations motivated by network configuration variations and weighted velocity strengths to increase search diversity. Specifically, the proposed model adopts a group-based encoding strategy to stimulate particle natural movement while guarding against invalid architectures. It also employs a novel particle distance computation strategy for calculating the differences between the current position of particle X and the global best g_{best} and personal best p_{best} solutions, respectively. Such a search strategy enables the swarm to thoroughly explore the search space between the particles and the local and global optimal signals in an attempt to identify the network configuration gaps, therefore increasing the chances of achieving global optimality. To increase diversification, a new velocity updating mechanism is adopted to randomly select the layers from either the distance between X and g_{best}, or the distance between X and p_{best}. Moreover, the proposed model weights the strength of velocity updates to implement a granular movement to balance between exploration and exploitation.

The pseudo-code of the proposed PSO algorithm for deep architecture generation is illustrated in Algorithm 1. Firstly, the training and test sets are obtained for each image database. A swarm of particles is initialized, where the position of each particle presents a potential network architecture. The proposed PSO algorithm is used to evolve the architecture and parameter settings of deep networks based on the proposed search operations. Specifically, the particles explore the search space by using a new weighted position updating procedure. During fitness evaluation, each particle encoded position is converted into a CNN model, which is subsequently trained using the training data set. The average training entropy loss is used as the fitness score to update those of p_{best} and g_{best} accordingly, if the current solution is fitter. The best architecture is obtained based on the global best position identified during the search process. It is then fully trained using the training set with a comparatively larger training epoch, and tested using the unseen test set. Each key proposed component is covered in detail within the following subsections.

4.2.1 The Proposed Encoding Strategy

A search space consists of all possible combinations of available settings including the number of layers and layer configurations. A particle represents one instance of a particular set of encoded settings which are used to describe a model architecture. The proposed encoding strategy adopts a group-based structure for describing a network, as shown in Fig. 25.

The underlying rationale of the group-based encoding strategy is as follows. It is designed by embedding human knowledge to ensure that the convolutional layers will always be followed by optional pooling layers. The number of pooling layers can be adjusted in accordance with the input image size. In other words, the encoding process ensures that the position of the pooling layers and the frequency of pooling operations will be valid, corresponding to the input image size. It simplifies implementation and does not artificially disrupt the natural particle movement.

Specifically, a group contains a number of convolutional layers and an optional pooling layer. A network contains multiple groups to vary down-sampling, but is limited by g_{max} to ensure down-sampling does not occur too frequently. The final group in a network is always followed by a fully connected layer for classification. This proposed group-based deep network generation strategy ensures all the formulated CNN models are valid whilst still providing sufficient flexibility for search space exploration without the requirement to specify the additional guarding rules.

In addition to the layer type, hyperparameter meta-data are encoded. In the case of a convolutional layer, the kernel size is encoded as $\{k \in \mathbb{R} | k_{min} \leq k \leq k_{max}\}$, and the number

Algorithm 1 The proposed PSO-based deep architecture generation model

```
1:  procedure PSO-BASED CNN MODEL GENERATION
2:      Initialize training and test data sets
3:      Initialize a swarm population
4:      for (each $t$ iteration) do
5:          for (each particle $X$ in swarm) do
6:              Construct a new model based on the current particle position
7:              $loss \leftarrow trainModel()$
8:              if $loss < p_{best}.fitness$ then
9:                  $p_{best}.fitness \leftarrow loss$
10:                 Update the personal best position
11:             end if
12:             if $loss < g_{best}.fitness$ then
13:                 $g_{best}.fitness \leftarrow loss$
14:                 Update the global best position
15:             end if
16:             Update the particle position by using the proposed weighted position updating procedure
17:         end for
18:     end for
19:     Save $g_{best}$ and initialize the identified best model based on $g_{best}$
20:     Train the final network using the training set and a larger training epoch
21:     Test the final model with the unseen test set
22:     Output the classification error rate
23: end procedure
```

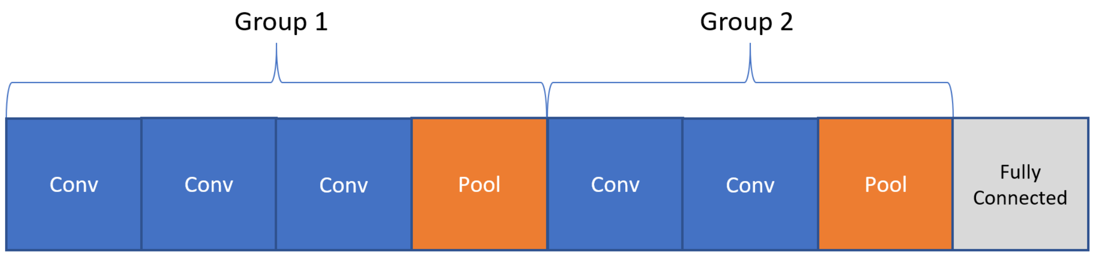


Figure 25: An example model containing two groups where each group contains convolutional layers and an optional final pooling layer

of output channels as $\{c_{out} \in \mathbb{R} | out_{min} \leq c_{out} \leq out_{max}\}$. Pertaining to the pooling layers, the pooling type is encoded as $\{p_{type} \in \mathbb{R} | 0 \leq p_{type} \leq 1\}$. The type of the pooling layer is selected according to the value of p_{type}. The types of pooling include the max and average poolings, or none in which case pooling is skipped. Different types of pooling layers are assigned based on a pre-determined threshold setting (see Section 4.3 for detail).

The proposed algorithm encodes the initial network depth as $\{d \in \mathbb{R} | 1 \leq d \leq d_{max}\}$ which

Table 15: The optimized network parameters and their corresponding search ranges. The settings of the search ranges adopted in the experiments are detailed in Section 4.3.

Layer	Parameter	Range
Convolution	Kernel k	$\{k \in \mathbb{R} \mid k_{min} \leq k \leq k_{max}\}$
	Number of channels c_{out}	$\{c_{out} \in \mathbb{R} \mid out_{min} \leq c_{out} \leq out_{max}\}$
Pooling	Pooling type p_{type}	$\{p_{type} \in \mathbb{R} \mid 0 \leq p_{type} \leq 1\}$
Depth	Number of layers l	Automatically optimized during the velocity update evol

is subsequently split into groups. A network contains g groups where $\{g \in \mathbb{R} \mid 1 \leq g \leq g_{max}\}$. The number of convolutional layers in each group is initialized by setting $l_{initial}$ using Equation 18 so that the initial number of layers is evenly distributed between groups. During the evolving process, the number of convolutional layers for each group is further optimized. The final layer of a group is always a pooling layer, where the pooling type or whether pooling occurs is determined by the current value of p_{type}. As down-sampling halves the input dimension size, another key advantage of the proposed group-based method is that g_{max} can be set to reflect the dimension of the input images, thus ensuring down-sampling is omitted in situations that can cause the model to become invalid, mitigating the need for additional rules. In other words, the proposed approach ensures the position and maximum frequency of the pooling always result in a valid model architecture without the need for complex governing rules such as those imposed by psoCNN, which interrupt the natural particle movements and complicate the implementation. In this research, the number of groups is limited to 2, owing to the input image size (i.e. 28x28). However, for data sets with larger images, the number of groups can be increased accordingly.

$$l_{initial} = \frac{d}{g} \tag{18}$$

As the final group is always followed by a fully connected layer, the model maps the output channel in the final network layer c_{out} to the number of target classes n_{class} automatically. The parameters optimized by the proposed PSO algorithm including their search ranges are provided in Table 15.

4.2.2 Initialization

A swarm consists of N particles initialized with randomly assigned positions. The first step of initialization is to randomly set the numbers of groups g and depth d for each particle. For each group, l convolutional layers are initialized according to Equation 18 with one pooling layer. Initializing a convolutional layer is performed by randomly assigning the kernel size k and the number of output channel c_{out}, respectively. The value of p_{type} is also randomly selected for the pooling layer.

4.2.3 Fitness Evaluation

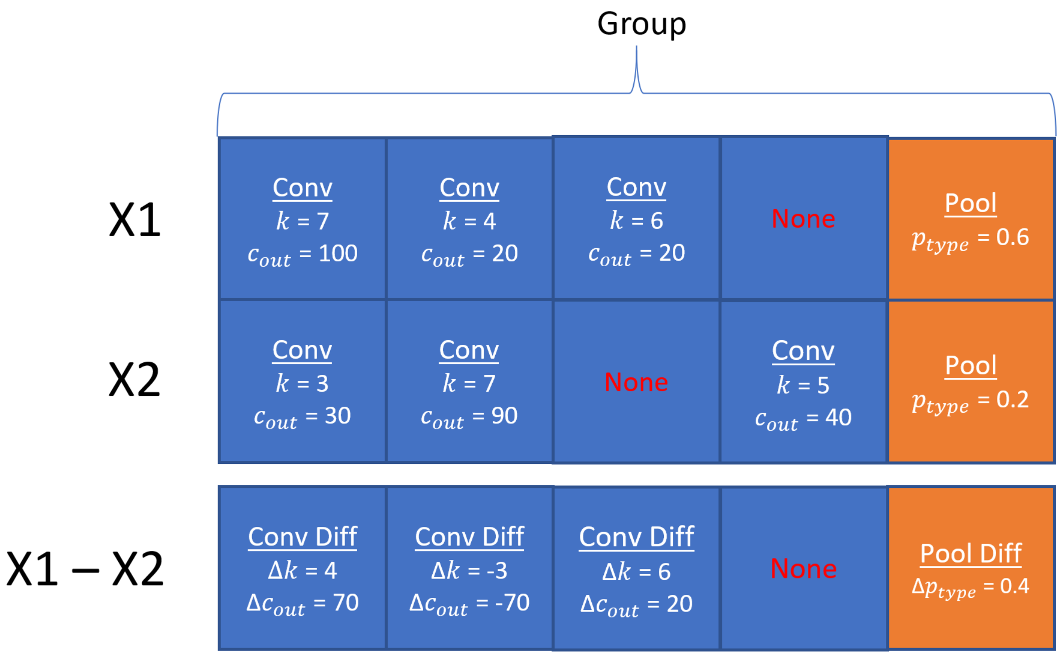
When evaluating the fitness of a particle, a new model is constructed based on the hyper-parameter settings for a given particle. The model is then trained on the training set for 1 epoch. Next, the average loss of the Adam optimizer [156] is computed during the training phase. This average training loss is used as the fitness score. The overall objective of the PSO algorithm during the optimization process is to minimise the average loss. The model with the most optimal network configuration is recommended as the global best solution. It is subsequently trained with a larger number of epochs using the training set and evaluated with the unseen test set for performance comparison.

4.2.4 Particle Distance Calculation

In the PSO algorithm, an individual particle X moves in the search space by following the personal and global best solutions. The calculation of the distance from the particle's personal best position p_{best}, and the distance from the global best position g_{best} is vital for search space exploration. The proposed position distance computation between two particles is introduced as follows.

Particles often have different lengths owing to different architecture configurations. As such, on a group-by-group basis, the shallower group is padded to the same length by temporarily appending the empty layers. Once both particle lengths match, the distance of particle $X2$ with respect to particle $X1$ defined as $X1 - X2$ is calculated depending on the layer types. For the convolutional layers, $X1 - X2$ is computed by subtracting the current values of k and c_{out} of $X2$ from those of $X1$ to return the distances represented as Δk and Δc_{out}. Two special cases exist. Specifically, if a convolutional layer of $X1$ is empty, the output is empty. Conversely, if a convolutional layer of $X2$ is empty, the configurations of the corresponding convolutional layer for $X1$ are copied. Pertaining to the pooling layer, $X1 - X2$ is computed by subtracting the current value of p_{type} of $X2$, from that of $X1$ to return the distance represented as Δp_{type}. Fig. 26 visually demonstrates the proposed distance computation mechanism between two particles, including the special and normal cases. The proposed particle difference computation mechanism is applied to the distance computation between the current particle and its personal and the global best solutions by replacing $X1$ with p_{best} and g_{best}, respectively.

In comparison with the existing studies such as psoCNN [10], where the distance between two particles is yielded by directly copying the layer configurations from p_{best} or g_{best}, the proposed movement mechanism identifies the configuration variations between two particles as indicated in Fig. 27 and effectively explores the search space between the current particle and the local and global best solutions to avoid stagnation. In other words, the proposed strategy is able to devise new layer configurations, instead of inheriting the existing layer structures from p_{best} and g_{best} directly in order to increase search diversity. Therefore, the resulting model is able to better explore the search space and attain global optimality. How these position differences are used with respect to the velocity calculation is explained in the next section.

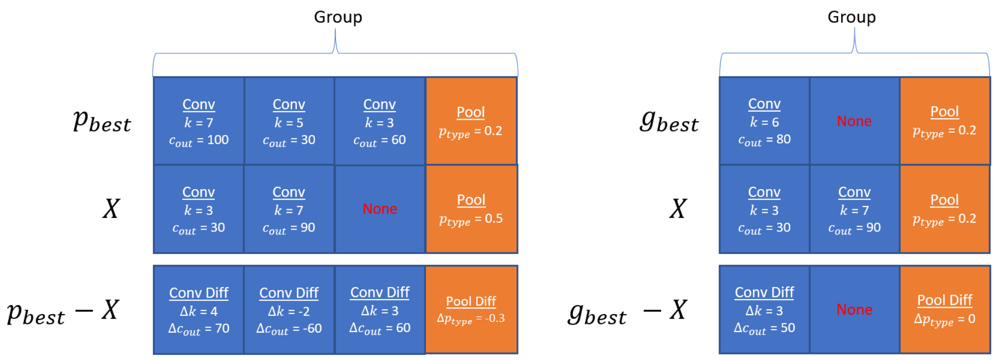


Group
X1
Conv k = 7 c_{out} = 100
Conv k = 4 c_{out} = 20
Conv k = 6 c_{out} = 20
None
Pool p_{type} = 0.6
X2
Conv k = 3 c_{out} = 30
Conv k = 7 c_{out} = 90
None
Conv k = 5 c_{out} = 40
Pool p_{type} = 0.2
X1 – X2
Conv Diff Δk = 4 Δc_{out} = 70
Conv Diff Δk = -3 Δc_{out} = -70
Conv Diff Δk = 6 Δc_{out} = 20
None
Pool Diff Δp_{type} = 0.4

Figure 26: Distance between particles calculated as $X1 - X2$

4.2.5 Velocity Calculation

To calculate velocity V, the distances of X with respect to p_{best} and g_{best} is calculated. Velocity is calculated for each group respectively using the proposed distance calculation mechanism. Fig. 27 illustrates an example for the distance calculation between the current particle and the personal and global best solutions.



Group
p_{best}
Conv k = 7 c_{out} = 100
Conv k = 5 c_{out} = 30
Conv k = 3 c_{out} = 60
Pool p_{type} = 0.2
X
Conv k = 3 c_{out} = 30
Conv k = 7 c_{out} = 90
None
Pool p_{type} = 0.5
$p_{best} - X$
Conv Diff Δk = 4 Δc_{out} = 70
Conv Diff Δk = -2 Δc_{out} = -60
Conv Diff Δk = 3 Δc_{out} = 60
Pool Diff Δp_{type} = -0.3
Group
g_{best}
Conv k = 6 c_{out} = 80
None
Pool p_{type} = 0.2
X
Conv k = 3 c_{out} = 30
Conv k = 7 c_{out} = 90
Pool p_{type} = 0.2
$g_{best} - X$
Conv Diff Δk = 3 Δc_{out} = 50
None
Pool Diff Δp_{type} = 0

Figure 27: Distance calculation between particle X and p_{best} and g_{best} respectively

Next, both resulting $g_{best} - X$ and $p_{best} - X$ distances are padded to the same length by temporarily appending the empty layers to the shallower group so that the same depth is achieved.

Next, for every m layer, a choice is made which decides whether to keep the resulting velocity calculation from $g_{best} - X$ or $p_{best} - X$ in the final velocity by generating a random number $\{r \in \mathbb{R} | 0 \leq r \leq 1\}$ and comparing it against a threshold α. The resulting velocity for each element is determined using Equation 19, as shown in Fig. 28, where g represents the group number and m denotes the number of layers in each group. In addition, α=0.5 was set to best match the setting of [10], in order to facilitate a direct comparison with the existing methods.

$$V_m^g = \begin{cases} p_{best\_m}^g - X_m^g & \text{if } r \leq \alpha, \\ g_{best\_m}^g - X_m^g & \text{otherwise} \end{cases} \tag{19}$$

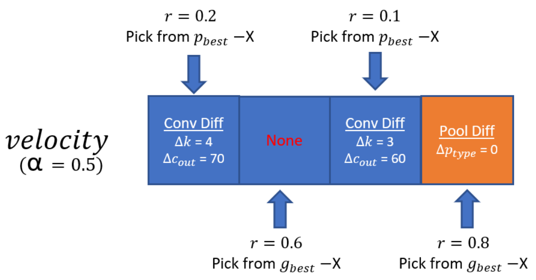


Figure 28: Calculating the final velocity by picking at random from $p_{best} - X$ or $g_{best} - X$

4.2.6 Particle Update

Velocity V obtained from the process is subsequently used to update the position of particle X. To facilitate a thorough exploration of the search space as well as increase the likelihood of generating more diversified layer configurations, a weighted velocity strength for position updating is used. Specifically, unlike the original PSO algorithm where the full velocity is used for position updating, as indicated in Equation 6, a weighting factor β is used to apply partial velocity for new position generation in the $(t+1)$-th iteration. The proposed position updating formula is defined in Equation 20.

$$X^{t+1} = \beta V + X^t \tag{20}$$

where β is the weighting factor used to control the degree at which the position of a particle is changed with respect to the velocity. The setting of β=0.5 in Equation 20 was selected based on trial-and-error. Moreover, position updates with respect to the kernel size k are bound purely by k_{min}. Likewise, position updates with respect to the number of channels c_{out} are bound only by out_{min}, in order to ensure the values remain valid. Such a position updating mechanism provides a granular and thorough exploration of the search space to increase the likelihood of finding global optimality and avoiding being trapped in local optima.

4.3 Experimental Studies

In these experimental studies, a swarm of 20 particles is initialized. Particle positions are updated over a maximum of 10 iterations. Full details of the settings used to constrain the search space when identifying the optimal network configurations are provided in Table 16.

The proposed model is implemented using PyTorch v1.5 [158]. During the optimization process, each prospective model represented by a particle is trained for 1 epoch with a mini-batch size of 64. Cross-entropy loss as the fitness criterion and Adam as the optimizer with a learning rate of 0.001 was selected for fair comparison as these are the settings adopted by closely related studies such as psoCNN [10]. The final best model indicated by the global best solution is re-trained for 100 epochs using the training set, which is subsequently evaluated using the test set.

All convolutions are performed with the same convolution formulation where padding is applied to the input to ensure matching between the input and output dimensions. Each convolution within the CNN, except for the first one, is preceded with a dropout layer with a dropout probability of 0.5. A convolutional layer is always followed by batch normalisation [154] and a ReLu activation function [159]. For the pooling operations, a non-overlapping 2x2 pooling mechanism is adopted. The final layer of each yielded model is a fully connected layer, which maps the outputs of the final convolutional layer to the number of classes of the respective data set.

Table 16: Algorithm settings and the search space used in these experiments. The settings selection was made to closely match those of existing studies [10] so that a fair comparison can be made.

Name	Description	Value Used
k_{min}	Minimum kernel size	3
k_{max}	Maximum kernel size	7
out_{min}	Minimum number of channels	3
out_{max}	Maximum number of channels	256
d_{max}	Maximum depth	20
g_{max}	Maximum number of groups	2
α	Layer selection boundary threshold	0.5
β	Weighting factor	0.5

4.3.1 Algorithm Parameter Settings

The settings shown in Table 16 are used for all experiments. The values and constraints in Table 16 are manually selected to best match those proposed in [10], in order to facilitate direct comparison between the competing methods. As an example, the α parameter in Equation 19 is used as the threshold to determine if each dimension of the new velocity is generated using the position difference from the personal or global best solution. The setting for α=0.5 was chosen to best match the setting adopted by psoCNN [10] to ensure a fair comparison. Such a setting (i.e. α=0.5) gives an equal consideration of the position difference from both best solutions as a reasonable trade-off. In these experiments, the setting for β=0.5 in Equation 20 was chosen by performing a grid search at 0.1 intervals. β is the weight factor used to control the degree at which the position of a particle is changed with respect to the velocity. As the benchmark data sets used have a size of 28x28, a setting of $g_{max} = 2$ was selected so that down-sampling does not cause the dimension to reduce below 8x8. Moreover, the pooling parameter, p_{type}, is optimized during the optimization process as shown in Table 15. It has a search range of [0, 1]. Equation 24 is used to define the pooling type according to the optimized value of p_{type}.

$$pooling = \begin{cases} NoPooling & \text{if } p_{type} \leq 0.33, \\ AvePooling & \text{if } p_{type} > 0.33 \ \& \ \ p_{type} \leq 0.66, \\ MaxPooling & \text{otherwise} \end{cases} \tag{21}$$

4.3.2 Benchmark Models

A number of hand-crafted networks and deep architecture generation methods are employed for performance comparison. In particular, several PSO-based algorithms are used in the experiments. As an example, IPPSO [13] is adopted which employs a test methodology consisting of 20 particles over 10 iterations constrained to a maximum of 9 convolutional layers and 3 fully connected layers.

Another state-of-the-art PSO-based architecture generation method, i.e. psoCNN [10], is also selected for comparison. It adopts a test methodology consisting of 20 particles optimized over 10 iterations. The search space constraints include a kernel size between 3x3 and 7x7 inclusive, a maximum of 256 channels and an upper limit of 20 layers comprising a combination of convolutional, pooling and fully connected layers. During the training phase, each candidate model is trained for 1 epoch using the Adam optimizer, where the swarm objective is to minimise the average loss.

In addition to PSO-based approaches, a GA-based called EvoCNN [12] is selected for performance comparison. EvoCNN employs the selection, crossover and mutation operators for CNN architecture generation. It generates offspring chromosomes from an initial parent pool size of 100. The variable-length gene encoding strategy is used to represent the CNN architectures with diverse lengths where convolutional, pooling and fully connected layers are embedded.

Moreover, MetaQNN [11] is used for performance comparison. It is a meta-modeling algorithm based on reinforcement learning for generating high-performing CNN architectures. Finally, three variants of the LeNet model [1] are also adopted as examples of hand-crafted networks for performance comparison. The LeNet models include LeNet-1 which contains twelve convolutional layers, two average pooling layers and one linear layer, LeNet-4 which contains twenty convolutional layers, two average pooling layers and two linear layers, and LeNet-5 which contains twenty-two convolutional layers, two average pooling layers and three linear layers.

4.3.3 Data Sets

A total of eight well-known benchmark data sets were selected for performance comparison, namely Rectangles [160], Rectangles-I [160] which consists of rectangles against image backgrounds, Convex [160], MNIST [1], and four MNIST variant data sets. These data sets are selected to aid direct comparison between the proposed method and the aforementioned recent models. The MNIST data set [1] contains images of handwritten digits ranging from 0 to 9, with size-normalized and centered. The data set consists of 60,000 training and 10,000 test samples. The MNIST variant data sets are comparatively more challenging than MNIST because of the additional distraction factors. As an example, MNIST-RD [160] comprises rotated MNIST digits, while MNIST-RB [160] contains MNIST digits with random background. MNIST-BI [160] consists of MNIST digits against background images, while MNIST-RD+BI [160] contains rotated MNIST digits against background images. Table 17 provides a summary of the aforementioned data sets.

Table 17: A summary of the data sets used during the experiments, all of which have an input size of 28 x 28 x 1

Data set	Description	Classes	Train/Test Samples
MNIST	Handwritten digits	10	60,000/10,000
MNIST-RD	MNIST with rotated digits	10	12,000/50,000
MNIST-RB	MNIST with random backgrounds	10	12,000/50,000
MNIST-BI	MNIST with background images	10	12,000/50,000
MNIST-RD+BI	MNIST with rotated digits and background images	10	12,000/50,000
Rectangles	Rectangle border shapes	2	12,000/50,000
Rectangles-I	Rectangle border shapes against image backgrounds	2	12,000/50,000
Convex	Convex shapes	2	8,000/50,000

4.3.4 Results

4.3.5 Performance Comparison with Existing Studies

Table 18 presents the experimental results for eight data sets. For the three LeNet models, i.e. LeNet-1, LeNet-4 and LeNet-5, the results for the MNIST data set are taken from the original publication [1]. Experiments are conducted using the three LeNet models for the remaining data sets and present the results in Table 18. For all other benchmark methods, i.e. MetaQNN [11], EvoCNN [12], IPPSO [13] and psoCNN [10], the results reported in their original studies are provided, to ensure a fair comparison.

The last two rows in Table 18 present the mean classification error rate achieved by the proposed method over 10 runs, as well as the best result from the 10 runs, for each data set. The remaining rows are the mean and best error rates reported by the compared methods. The results in which the proposed method outperforms the competitors are highlighted in bold. As illustrated in Table 18, the proposed approach achieves a superior performance, indicated by a reduction in the error rates reported across nearly all the test data sets in comparison with all the baseline models, with the same swarm size, iterations and constrains. In addition, psoCNN is the best performing baseline method across all data sets. In Table 19, the error rates of the proposed model is compared against those of psoCNN [10], to clearly indicate performance improvements.

Table 18: Experimental results compared against various benchmark methods in terms of error rates. Results in bold indicate a reduction in error rate when compared with the benchmark methods. For the LeNet models, the results reported for MNIST are taken from the original study [1]. The LeNet model results for the remaining data sets were obtained by training the model from scratch. The results of MetaQNN, EvoCNN, IPPSO and psoCNN are extracted from their original studies, i.e. [11], [12], [13] and [10], respectively.

Model	MNIST	MNIST-RD	MNIST-RB	MNIST-BI	MNIST-RD+BI	Rectangles	Rectangles-I	Convex
Hand-crafted architectures								
LeNet-1	1.70% [1]	19.3%	7.50%	9.80%	40.06%	0.08%	16.92%	10.61%
LeNet-4	1.10% [1]	11.79%	6.18%	8.96%	33.83%	0.05%	16.09%	8.39%
LeNet-5	0.95% [1]	11.10%	5.99%	8.70%	34.64%	0.07%	12.48%	8.40%
Reinforcement learning techniques								
MetaQNN (best) [11]	0.44%	-	-	-	-	-	-	-
Evolutionary optimization techniques								
EvoCNN (best) [12]	1.18%	5.22%	2.80%	4.53%	35.03%	0.01%	5.03%	4.82%
EvoCNN (mean) [12]	1.28%	5.46%	3.59%	4.62%	37.38%	0.01%	5.97%	5.39%
IPPSO (best) [13]	1.13%	-	-	-	33%	-	-	8.48%
IPPSO (mean) [13]	1.21%	-	-	-	34.50%	-	-	12.06%
psoCNN (best) [10]	0.32%	3.58%	1.79%	1.90%	14.28%	0.03%	2.22%	1.70%
psoCNN (mean) [10]	0.44%	6.42%	2.53%	2.40%	20.98%	0.34%	3.94%	3.90%
The proposed system								
ours (best)	0.35%	**3.23%**	1.8%	2.2%	**11.61%**	**0%**	**1.01%**	**1.36%**
ours (mean)	**0.38%**	**3.9%**	**1.88%**	2.46%	**13.4%**	**0%**	**1.57%**	**1.63%**

Table 19: The mean error rates over 10 runs for the proposed method and psoCNN [10], along with the performance differences between the two methods (where the (-) symbol indicates that the proposed model is better and the (+) symbol indicates that the proposed model is worse).

Model	MNIST	MNIST-RD	MNIST-RB	MNIST-BI	MNIST-RD+BI	Rectangles	Rectangles-I	Convex
ours (best)	0.35%	**3.23%**	1.8%	2.2%	**11.61%**	**0%**	**1.01%**	**1.36%**
ours (mean)	0.38%	3.9%	1.88%	2.46%	13.4%	0%	1.57%	1.63%
psoCNN (best) [10]	0.32%	3.58%	1.79%	1.90%	14.28%	0.03%	2.22%	1.70%
psoCNN (mean) [10]	0.44%	6.42%	2.53%	2.40%	20.98%	0.34%	3.94%	3.90%
error difference (best)	0.03%(+)	**0.35%(-)**	0.01%(+)	0.30%(+)	**2.67%(-)**	**0.03%(-)**	**1.21%(-)**	**0.34%(-)**
error difference (mean)	**0.06%(-)**	**2.52%(-)**	**0.65%(-)**	0.06%(+)	**7.58%(-)**	**0.34%(-)**	**2.37%(-)**	**2.27%(-)**

MNIST represents a relatively simple benchmark data set with limited room for improvement. The networks devised by the proposed method achieve a mean error rate of 0.38% for MNIST, which is an improvement of 0.06% as compared with those from the networks yielded by psoCNN. The best CNN model constructed with the proposed method achieves the best error rate of 0.35%, which is within a reasonable error margin as compared with the

Table 20: Result comparison between the proposed model using the encoding strategy only, and the proposed model using both encoding and search strategies, over 10 runs.

Model	MNIST	MNIST-RD	MNIST-RB	MNIST-BI	MNIST-RD+BI	Rectangles	Rectangles-I	Convex
psoCNN (best) [10]	0.32%	3.58%	1.79%	1.90%	14.28%	0.03%	2.22%	1.70%
psoCNN (mean) [10]	0.44%	6.42%	2.53%	2.40%	20.98%	0.34%	3.94%	3.90%
ours (best) encoding + copy	0.36%	3.32%	2.07%	2.44%	15.68%	0.02%	1.58%	1.47%
ours (mean) encoding + copy	0.42%	4.76%	2.25%	3.43%	18.00%	0.47%	2.30%	1.94%
ours (best)	0.35%	3.23%	1.8%	2.2%	11.61%	0%	1.01%	1.36%
ours (mean)	0.38%	3.9%	1.88%	2.46%	13.4%	0%	1.57%	1.63%

Table 21: The mean search time in minutes of the experiments using (1) purely the proposed encoding strategy, and (2) the overall proposed model for the training and search phase over 10 runs and their corresponding improvements against those of psoCNN. The (-) symbol indicates that the proposed strategies are better in computational costs. All experiments have been conducted using one NVIDIA GeForce RTX 2080Ti consumer GPU.

Model	MNIST	MNIST-RD	MNIST-RB	MNIST-BI	MNIST-RD+BI	Rectangles	Rectangles-I	Convex
psoCNN (mean) [10]	276	47	49	56	43	14	46	33
ours (mean) encoding + copy	268	43	44	29	29	7	42	30
Speed improvement encoding + copy	-3%	-9%	-10%	-48%	-33%	-49%	-8%	-6%
ours (mean)	192	22	23	21	26	5	33	29
Speed improvement	-30%	-53%	-53%	-63%	-40%	-61%	-29%	-11%

best error rate of 0.32% from psoCNN.

With respect to the MNIST-RD data set, the psoCNN method obtains a mean error rate of 6.42%. The proposed method constructed a network that produced a mean error rate of 3.9%, which illustrates an improvement of 2.52%. The devised best network achieves the best error rate of 3.23%, outperforming the best model from psoCNN by 0.35%.

Moreover, for MNIST-RB, again psoCNN is the leading baseline method with a mean error rate of 2.53%. The proposed method yields a network with a mean error rate of 1.88%, and outperforms psoCNN by 0.65%.

For the MNIST-BI data set, the devised networks achieve a mean error rate of 2.46%. The psoCNN model achieves a slightly better performance with a mean error rate of 2.4%.

For MNIST-RD+BI, the generated networks achieve a mean error rate of 13.4%, which shows a significant improvement of 7.58% over the mean result of 20.98% obtained by the networks devised by psoCNN. The identified best network also produces the best error rate of 11.61%, and outperforms that yielded by psoCNN by 2.67%.

For the Rectangles data set, the approach achieves the mean and best error rates of 0%. This indicates that the proposed approach is able to devise 10 CNN networks, all of which achieve a 100% accuracy rate. On the other hand, psoCNN and EvoCNN obtain the mean error rates of 0.34% and 0.01%, respectively.

For the Rectangles-I data set, the optimized networks achieve a mean error rate of 1.57%, and outperform those generated by psoCNN by 2.37%. Moreover, the identified best CNN model achieves the best error rate of 1.01%, which shows an improvement of 1.21% as compared with the best network yielded by psoCNN.

For the Convex data set, the generated CNN models achieve a mean error rate of 1.63%, which outperforms those produced by psoCNN by 2.27%.

A convergence curve comparison between the proposed model and psoCNN is provided. Fig. 29 plots the mean entropy loss scores of the g_{best} solutions for both methods in the training stage over 10 runs with respect to all data sets. The proposed model illustrates a faster reduction in loss and continuous improvements in subsequent iterations compared to psoCNN. As an example, for the MNIST-RD+BI data set, the mean loss of psoCNN flattens after iteration 4. Comparatively, the proposed model continues to achieve improvements until iteration 8. In short, the model shows faster convergence rates in comparison with those of psoCNN pertaining to the architecture search for all the data sets.

4.3.6 Effectiveness of the Proposed Encoding and Search Strategies

To further indicate the effectiveness of the proposed encoding and search strategies, experiments were conducted using purely the proposed encoding strategy as well as the overall proposed model. Specifically, Table 20 illustrates the mean results of the proposed model (1) with only the proposed encoding strategy, and (2) with both the proposed encoding and search mechanisms, over a set of 10 runs. The results from the proposed encoding strategy only are shown in rows 3 and 4 in Table 20. When evaluating the encoding strategy in isolation, the same copying strategy of psoCNN [10] was used as the search operations. Specifically, the layers are copied randomly from either the global or personal best position using the same decision boundary setting of $\alpha = 0.5$, as that used in psoCNN. When using the proposed encoding strategy only, the results indicate a reduction of the mean error rates for 6 out of 8 data sets, as compared with those of psoCNN, as presented in Table 20. Furthermore, the results indicate a further improvement across all 8 data sets when both the proposed encoding and search strategies are combined, as illustrated in rows 5 and 6 in Table 20. This clearly ascertains the capability of the proposed search mechanisms to enhance the performance over and above the improvements from adopting only the proposed encoding strategy.

4.3.7 Computational Cost Comparison

Table 21 provides computational cost comparison for the architecture search during the training stage between the proposed model and psoCNN over 10 runs. To indicate the effectiveness of the proposed strategies in reducing the computational cost, the table includes the mean search time (in minutes) using (1) solely the proposed encoding strategy, and (2) the overall proposed model. The proposed model and psoCNN adopt the same experimental settings, i.e., with the mean training entropy loss as the fitness score and a training epoch of one. In addition, the same swarm size of 20 is used, and the same maximum iteration cycle of 10 so that the search duration of all methods can be compared directly and fairly. All the experiments are conducted using one NVIDIA GeForce RTX 2080Ti consumer GPU.

When evaluating the encoding strategy only, the same copying search operation as that of psoCNN is used. In Table 21 the mean computational costs of the proposed model using purely the encoding strategy are shown in row 2 and its cost reduction against those of psoCNN in row 3. The proposed model with purely the encoding strategy illustrates enhanced computational efficiency in comparison with those of psoCNN for all data sets. This is owing to the initialization of the swarm with valid and comparatively reasonable network architectures using the proposed encoding strategy that is able to accelerate the search process. In addition, the mean computational costs of the proposed model are reported using both the proposed encoding and search strategies in row 4 and its cost reduction against those of psoCNN in row 5. The results indicate that the overall proposed model shows a greater computational cost reduction across all data sets, owing to the efficiency of the proposed search mechanisms for architecture evolution during the search process.

Because of the adoption of different fitness evaluation strategies, this study was unable to present a direct computational cost compared with other baseline methods, i.e. MetaQNN, EvoCNN and IPPSO. Specifically, these models train and test each optimized candidate network using the training and validation sets, respectively, which increases the search times due to the use of additional steps (which are computationally heavy). According to the original studies, MetaQNN [11] indicates a search time of 100 GPU days for the MNIST data set, EvoCNN [12] requires average 2-3 days using two GTX 1080 GPU cards for each data set, while IPPSO [13] consumes average 2.5 hours for the MNIST, MNIST-RD+BI and Convex data sets, respectively. The proposed model and psoCNN illustrate better computational efficiency in comparison with those of the aforementioned models in most of the test cases.

4.3.8 Discussion

4.3.9 Theoretical Justification

In this research, a group-based encoding strategy as well as new velocity and position updating mechanisms based on the key network configuration differences and weighted velocity strengths was proposed. The advantages of the proposed model in comparison with the most closely related one, i.e. psoCNN [10], are as follows.

The search process of the psoCNN model [10] generates a particular type of layer in any number (e.g. multiple convolutional or pooling layers). It also generates any combination of convolutional, pooling and fully connected layers for devising CNN architectures. Therefore, psoCNN requires additional governing rules to ensure the validity of the generated networks, which interrupts the exploration capability of the particles in the search space. In order to tackle such limitations, the proposed group-based encoding strategy ensures all the formulated CNN models are valid without the requirement of additional governing rules. Specifically, each group contains at least one convolutional layer, which is followed by an optional pooling layer. In other words, the pooling layer is always positioned as the final layer in the group. Moreover, the number of groups can be adjusted in accordance with the input image size. By restricting the number of groups, the frequency of pooling operations can be tailored toward the input image size. This ensures the position of pooling layers and the frequency of pooling operations are valid with respect to the input image size without the

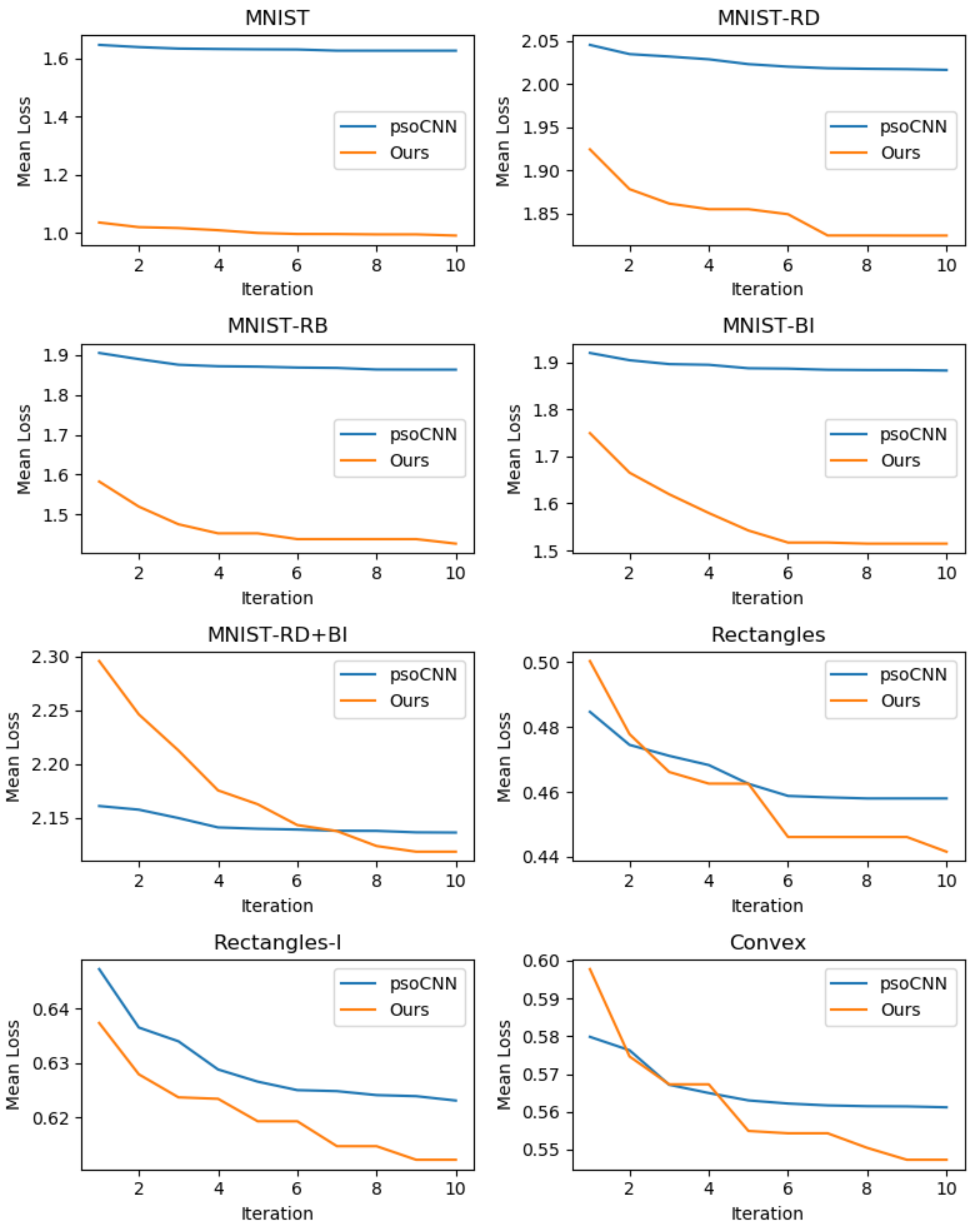


Figure 29: Convergence curves of the proposed algorithm and psoCNN. The mean losses of 10 runs are plotted over 10 iterations for all data sets

need for any additional governing rules. Instead of generating several fully connected layers as in psoCNN, the proposed model only attaches one fully connected layer as the last layer of the final group in a network, which is similar to the approach in [3].

Moreover, in psoCNN [10], instead of creating new layer configurations, the existing layer configurations from the personal and global best solutions are copied directly in the velocity updating operation (as illustrated in the right-hand side in Fig. 30). In contrast, the proposed model employs the key layer configuration differences between the current particle and the personal and global best solutions for new velocity generation (as illustrated in the left-hand in Fig. 30). Therefore, the proposed model is more capable of devising new network architectures using intermediate positions of the particles' trajectories. In addition, a new position updating mechanism that takes a partial strength of velocity updates to generate new positions was proposed. This provides the capabilities for the particles to explore the search space with various momentums and scales, in order to increase search diversity. Such a search mechanism allows a more granular exploration pertaining to the in-between positions, increasing the chances of devising more diversified networks. Therefore, the proposed model is less reliant on initialization, along with enhanced capabilities in evolving architecture generation.

4.3.10 Experimental Observations in Comparison with Related Studies

This section discusses how the proposed encoding strategy and search mechanisms lead to improvements in both speed and accuracy in comparison with those of psoCNN [10] through experimental observations. This section firstly explains the rationale of the proposed encoding strategy for enhancing both speed and accuracy. Then, a discussion is provided around the benefits of combining both the proposed search and encoding strategies further to improve the accuracy and speed of the final model.

To identify the contribution of the proposed encoding strategy in terms of computational efficiency, the encoding strategy is isolated from the proposed search operations by combining it with the copying search strategy used in psoCNN [10]. Next, the time taken to perform the architecture search in the training phase was recorded. The detailed computational costs are presented in Table 21. As discussed earlier, the results indicate that across all 8 data sets, the proposed encoding strategy improves the computational costs as compared with those of psoCNN [10]. Specifically, a reduction in computational costs in the architecture search stage by up to 49% on the Rectangles data set was observed. As indicated in Table 21, the proposed encoding strategy is the primary reason for the enhancement of computational efficiency. As an example, the encoding method of psoCNN allows the construction of models containing a large number of fully connected layers. In some cases, the models created by psoCNN contain 8 consecutive fully connected layers, resulting in a long training time. Subsequent iterations attempt to eliminate the additional fully connected layers and finally recommend a model with one fully connected layer towards the end of the search process. In contrast, as discussed above, the proposed encoding strategy avoids such a problem by fixing the number of fully connected layers to one, as well as fixing its position to the final layer of the model, therefore contributing toward the reduction in computational costs.

To identify the contribution of the proposed encoding strategy in terms of accuracy, Table 20 shows that by applying the proposed encoding strategy on its own and adopting the same velocity updating mechanism as that of psoCNN (i.e. the layer copying strategy from p_{best} or g_{best}), it results in an improvement in the mean accuracy rates (of 10 runs) pertaining to the MNIST, MNIST-RD, MNIST-RB, MNIST-RD+BI, Rectangles-I and Convex data sets. These empirical results reveal that psoCNN has constructed pooling layers one after another, leading to poor performing networks and wasteful function evaluations. The experimental results also indicate that the use of hardcoded rules in psoCNN for ensuring model validity interrupts the natural particle movement. This is ascertained by the aforementioned accuracy improvements when applying the proposed encoding strategy alone. The proposed encoding strategy ensures that undesirable candidate models such as those containing stacks of pooling layers are never constructed since the pooling position is determined by a group-based structure in the proposed scheme. Each group can only contain at most one pooling layer, which is always positioned as the final layer in the group after at least one convolutional layer. Pooling layers, therefore, are never stacked. Furthermore, rules that can interrupt the natural particle movement such as eliminating excessive pooling layers are not required in the proposed scheme, as the maximum number of pooling layers is determined by the maximum number of groups. Therefore, the proposed model eliminates the possibility of having excessive down-sampling operations that can cause the network to become invalid, mitigating the need for additional governing rules. Indeed, the proposed encoding strategy ensures that each devised model is reasonably constructed that helps exploit the search space in a comprehensive manner.

As indicated in Table 20 and Table 21, combining the proposed search mechanisms (both velocity and position updating operations) and the proposed encoding strategy together results in further improvement of accuracy and computational efficiency as compared with those yielded by using the proposed encoding strategy alone. By combining both strategies together, the mean accuracy rate increases by up to 7.58% pertaining to the MNIST-RD+BI data set and the mean computational cost reduces by up to 63% pertaining to the MNIST-BI data set. This suggests that the copying search strategy adopted by psoCNN is less flexible because such a strategy heavily relies on a good initialization. It does not create new layer structures for new velocity generation but simply copies the existing layer configurations from either the p_{best} or g_{best} solutions, as illustrated in the right-hand part of Fig. 30. Comparatively, the proposed velocity updating mechanism combined with a granular weighted movement operation is able to formulate completely new particle solutions by identifying the key layer configuration differences between particles, as shown in the left-hand part of Fig. 30. In other words, the proposed search strategies are able to yield new layer configurations which cannot be achieved by the copying operation of psoCNN. In addition, owing to the capabilities of searching the intermediary positions, the proposed velocity and granular weighted position updating mechanisms have better diversification and intensification, leading to better devised networks, as ascertained by the empirical results. In short, both proposed encoding and search mechanisms illustrate superior efficiency in terms of speed and performance in comparison with those of psoCNN.

The proposed model and psoCNN are significantly faster than reinforcement learning and other evolutionary optimization methods for deep architecture generation, whilst not sacrificing accuracy. The search cost of the proposed model consumes 5 to 192 minutes, while that of psoCNN consumes 14 to 276 minutes, for all the test data sets. On the other hand, MetaQNN [11], which is based on reinforcement learning, requires 100 GPU days for processing the MNIST data set, while EvoCNN [12], which uses a GA-based method, needs

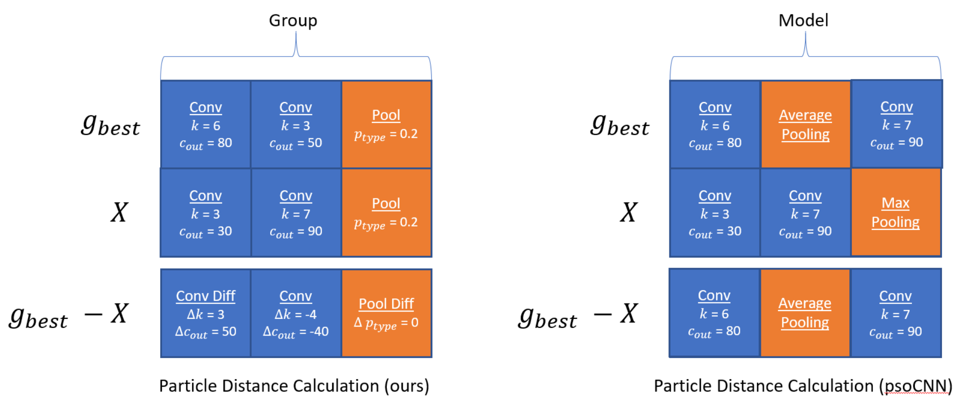

Group
Model
g_{best}
X
$g_{best} - X$
Conv k = 6 c_{out} = 80
Conv k = 3 c_{out} = 50
Pool p_{type} = 0.2
Conv k = 3 c_{out} = 30
Conv k = 7 c_{out} = 90
Pool p_{type} = 0.2
Conv Diff Δk = 3 Δc_{out} = 50
Conv Δk = -4 Δc_{out} = -40
Pool Diff Δp_{type} = 0
Particle Distance Calculation (ours)
Conv k = 6 c_{out} = 80
Average Pooling
Conv k = 7 c_{out} = 90
Conv k = 3 c_{out} = 30
Conv k = 7 c_{out} = 90
Max Pooling
Conv k = 6 c_{out} = 80
Average Pooling
Conv k = 7 c_{out} = 90
Particle Distance Calculation (psoCNN)

Figure 30: Comparison of particle distance calculations between the proposed approach and the psoCNN method

on average 2-3 days of processing time with two GTX1080 GPU cards for each data set.

4.3.11 Discussion of Identified Models

The best CNN models identified by the proposed method for all the test data sets are shown in Table 22. In some instances, such as on the Rectangles data set, numerous models that achieve the same best performance (i.e. 0% error rate) are identified. While the computational cost is not an objective in the fitness function, significantly smaller models are generated in this study as compared with those reported by psoCNN. As an example, the best model contains two convolutional layers with 70 and 60 channels, respectively, for the Rectangles data set, whereas the network produced by psoCNN contains three convolutional layers with 139, 113, and 226 channels respectively. Average pooling is the most common method of pooling identified in the experiments, which is the same observation found in related studies. The empirical results indicate that performance can be further improved by using more iterations and larger population sizes.

Table 22: The best CNN models discovered by the proposed approach for the eight test data sets

Data set	Group	Layer Type	Layer Setting
MNIST	1	Convolution	$k = 5$, $c_{out} = 165$
	1	Convolution	$k = 6$, $c_{out} = 220$
	1	Convolution	$k = 5$, $c_{out} = 215$
	1	Pooling	Average Pooling
	2	Convolution	$k = 4$, $c_{out} = 120$
	2	Pooling	Average Pooling
MNIST-RD	1	Convolution	$k = 6$, $c_{out} = 66$
	1	Convolution	$k = 5$, $c_{out} = 140$
	1	Convolution	$k = 6$, $c_{out} = 192$
	1	Pooling	Average Pooling
	2	Convolution	$k = 7$, $c_{out} = 217$
	2	Pooling	Average Pooling
MNIST-RB	1	Convolution	$k = 4$, $c_{out} = 230$
	1	Convolution	$k = 6$, $c_{out} = 109$
	1	Convolution	$k = 4$, $c_{out} = 219$
	1	Convolution	$k = 4$, $c_{out} = 23$
	1	Convolution	$k = 5$, $c_{out} = 112$
	1	Convolution	$k = 7$, $c_{out} = 174$
	1	Convolution	$k = 5$, $c_{out} = 36$
	1	Pooling	Average Pooling
MNIST-BI	1	Convolution	$k = 4$, $c_{out} = 153$
	1	Convolution	$k = 4$, $c_{out} = 153$
	1	Convolution	$k = 7$, $c_{out} = 159$
	1	Convolution	$k = 7$, $c_{out} = 195$
	1	Convolution	$k = 7$, $c_{out} = 18$
	1	Pooling	Average Pooling
MNIST-RD+BI	1	Convolution	$k = 4$, $c_{out} = 170$
	1	Convolution	$k = 4$, $c_{out} = 247$
	1	Convolution	$k = 3$, $c_{out} = 200$
	2	Convolution	$k = 5$, $c_{out} = 136$
	2	Convolution	$k = 6$, $c_{out} = 137$
	2	Convolution	$k = 6$, $c_{out} = 38$
	2	Pooling	Average Pooling
RECTANGLES	1	Convolution	$k = 7$, $c_{out} = 70$
	1	Convolution	$k = 7$, $c_{out} = 60$
	1	Pooling	Average Pooling
RECTANGLES-I	1	Convolution	$k = 4$, $c_{out} = 204$
	1	Convolution	$k = 5$, $c_{out} = 202$
	1	Convolution	$k = 5$, $c_{out} = 210$
	1	Convolution	$k = 5$, $c_{out} = 44$
	1	Convolution	$k = 4$, $c_{out} = 49$
	1	Pooling	Average Pooling
	2	Convolution	$k = 4$, $c_{out} = 141$
	2	Convolution	$k = 4$, $c_{out} = 176$
	2	Convolution	$k = 5$, $c_{out} = 41$
	2	Pooling	Average Pooling
CONVEX	1	Convolution	$k = 4$, $c_{out} = 193$
	1	Convolution	$k = 5$, $c_{out} = 225$
	1	Convolution	$k = 5$, $c_{out} = 108$
	1	Convolution	$k = 5$, $c_{out} = 222$
	2	Convolution	$k = 7$, $c_{out} = 171$
	2	Convolution	$k = 5$, $c_{out} = 184$
	2	Convolution	$k = 7$, $c_{out} = 145$
	2	Convolution	$k = 6$, $c_{out} = 31$
	2	Pooling	Average Pooling

5 Proposed PSO Based Architecture Generation with Residual Connections for Image Classification

5.1 Introduction

The motivation of this research is to address the following weaknesses in the state-of-the-art CNN architecture construction research. 1) Existing methods that perform automatic CNN generation do not exploit residual connections. Residual connections are a recent technique that enables deeper model constructions by avoiding the vanishing gradient problem. Residual connections provide an alternative path for data to flow by skipping layers. Residual connections skip layers by connecting the output of a convolutional layer to the input of a layer deeper in the network through an add operation. As skip connections establish a shorter path between the first and final layers of the model, when gradients backpropagate through the network during training, they are less likely to approach zero. Avoiding gradients approaching zero helps prevent the vanishing gradient problem so that a deeper network construction is possible. Such approaches that do not adopt residual connections are therefore limited with respect to model depth. 2) CNN optimization techniques perform a limited search on a base CNN model that may contain residual connections. Such optimization techniques do not consider the optimization of parameters like kernel size or pooling types. Such a limited search reduces the variety of the generated networks. 3) Existing PSO based CNN generation techniques may become trapped in local optima due to following the personal and global best leaders.

This research proposed the following novel contributions. 1) A new encoding strategy and velocity updating mechanism is proposed. The proposed strategy is capable of constructing deep CNN architectures comprising residual connections. This proposed approach maintains the ability to devise varied CNN architectures by automatically constructing network architectures composed of different kernel sizes, pooling types, depths and widths. 2) A novel search strategy that follows a non uniformly selected neighbouring best position and global best position is proposed. 3) Both proposed strategies are evaluated against existing state-of-the-art research on a range of data sets. Fig 31 illustrates the proposed system architecture.

5.2 The Proposed PSO-based Deep Architecture Generation

In this research, a PSO-based deep architecture generation method, namely resPsoCnn, to devise deep networks with residual connections is proposed. The proposed PSO algorithm incorporates a new encoding scheme and a new search mechanism guided by neighbouring and global promising solutions for deep architecture search. Specifically, the proposed encoding scheme is capable of representing deep CNN models comprising residual connections. The important settings, such as the numbers of groups and residual blocks, the kernel size and number of filters for each residual block, as well as pooling layer choices for each

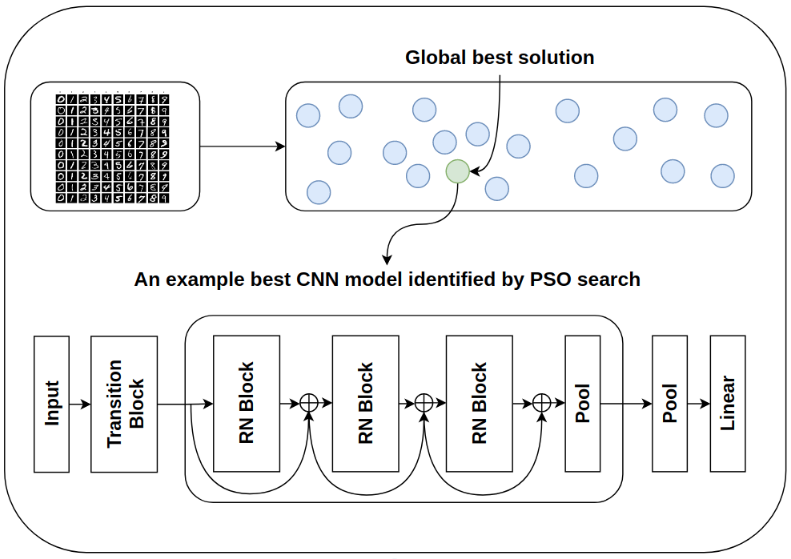


Figure 31: The proposed system architecture where the identified best model is indicated by the global best solution

group, are optimized using the proposed PSO algorithm. Moreover, the search process led by the swarm leader and non-uniformly randomly selected neighbouring promising solutions at the fine-grained and global levels illustrates a better balance between diversification and exploitation and produces a rich assortment of residual architectures with great diversity. Each key element of resPsoCnn is introduced in the following sub-sections.

5.2.1 Encoding Strategy and Initialization

The proposed encoding strategy stores multi-dimensional swarm position information for representing CNN model architecture configurations. At the start of the optimization process, a swarm containing a fixed number of particles is initialized with random particle positions constrained by predetermined search ranges. The following elements summarize the CNN architecture settings encoded within the proposed encoding strategy.

- A model contains at least one group. The number of groups is optimized between 1 and g_{max}.
- A group contains at least one residual block. The number of blocks the model can contain during initialization is set between 1 and b_{max}. The number of residual blocks in each group is optimized.
- All blocks within a group share the same number of channels for compatibility. The number of channels used by a group between out_{min} and out_{max} is optimized.
- A group contains an optional pooling layer, which can be of type max pooling, average pooling or no pooling. The pooling type is optimized by dividing a search range between 0 and 1 into three regions and attributing a pooling type to each region.

- A block contains a stack of convolutions layers, performing same convolutions, i.e. the appropriate padding is used to ensure the dimensions of the output match those of the input volume. The degree of padding depends on the kernel size. The kernel size of a convolutional layer is optimized on a block-by-block basis between k_{min} and k_{max}. This is necessary as the kernel size controls the receptive field, which in turn controls the visibility degree of an image with respect to one filter, at one time. [27].

The parameters optimized by the proposed PSO algorithm, including their search ranges, are summarized in Table 23.

Table 23: The optimized network parameters and their corresponding search ranges. The settings of the search ranges used in the conducted experiments are provided in Section 5.3.

Domain	**Parameter**	**Range**
Model	Number of groups	1 to g_{max}
Group	Number of residual blocks	1 to b_{max}
Group	The number of channels c_{out} for all blocks in a group	out_{min} to out_{max}
Convolution	Kernel size k	k_{min} to k_{max}
Pooling	Pooling type p_{type}	0 to 1

5.2.2 Decoding Strategy

The position information encoded within a particle is decoded to construct a valid CNN model architecture. As an example, a high level overview of a constructed CNN model after the decoding process is visualized in Fig. 32.

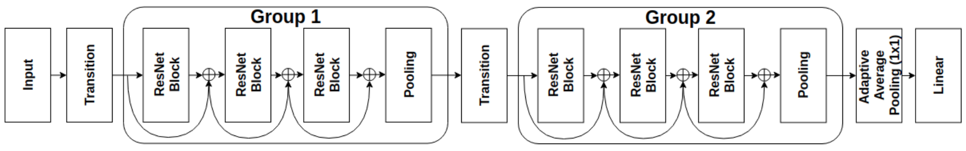


Figure 32: An example decoded network where the model configurations, i.e. the number of groups, the number of blocks per group, and the contents of each group (e.g. the kernel size of each ResNet block, the number of channels and the pooling type for each group), are embedded in the encoding process.

Residual connections require the number of output channels from the previous layer to match that of the current layer so that an add operation can be performed. A transition layer that precedes each group is used to either increase or decrease the number of output channels so that the dimension of the input fed into a group matches the expected dimension of the group. A transition block comprises a 1x1 convolution and a ReLU activation function. A ResNet block comprises two stacked sets of single convolutional layers, followed by a batch normalization layer and a ReLU layer. Both the transition block and the ResNet block are visualised in Fig. 33.

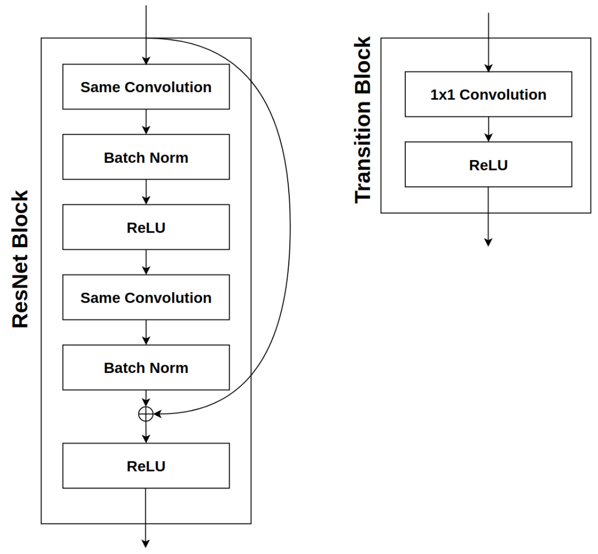


Figure 33: The structures of a ResNet block (left) and a transition block (right)

In addition, the final group of a model is followed by an adaptive average pooling layer and a linear layer. The linear layer performs the final image classification with the number of neurons set as the number of target classes in the data set.

Table 24 introduces the notations used in the following sections to represent particle encoded information such as the kernel size k of a particular block b, within the group g of a particle position X.

5.2.3 Optimization Strategy

The search for optimal architectures is conducted by optimizing important hyper parameters such as the kernel size and model depth. A list of the hyperparameters to be optimized is provided in Table 23. The optimization process is outlined in the following steps. In step 1, the particle distance is calculated in respect to the global and neighbouring best solutions as indicated in Section 5.2.4. In step 2, a new velocity based on the distances between the current particle position, and the global and neighbouring best solutions is calculated. This step is covered in more detail in Section 5.2.9. In step 3, the calculated velocity is combined with the current particle position to perform a particle position update as defined in Section 5.2.10. The fitness of a new particle is then evaluated using the fitness function, as defined in Section 5.2.11. The above search process iterates until the termination criterion is fulfilled. Table 24 introduces the notation used in the subsequent subsections of this research.

Table 24: A summary of the notations

Notation	**Description**
X_i	The position X of the ith particle in the swarm
g_n	The nth group
b_m	The mth block
$k_(X_i g_n b_m)$	Kernel size for the mth block of the nth group of the ith particle in position X

5.2.4 Particle Distance Calculation

During the optimization process, the generation of new particle velocity is an important step. The new velocity guides the particle movement by considering the distances between the best known positions, namely n_{best} and g_{best}, and the current particle position. In this research, a velocity update rule for guiding particle search is proposed. Such a velocity update calculation requires a mechanism for calculating the distance between a pair of particles. For this purpose, a new particle distance calculation method is also proposed. At a high level, the distance of X_2 with respect to X_1 is calculated as $X_1 - X_2$. This means that the resulting distance could be negative or positive, where the former indicates that a setting should be reduced and the latter indicates that a setting should be increased. The proposed particle distance calculation method, as well as the new velocity and position updating formulae are covered in the following sub-sections.

5.2.5 Distance Calculation between Groups with respect to the Number of Channels c_{out}

The particle distance is calculated on a group-by-group basis. A group maintains a record of the number of output channels c_{out} that each block within the group should use. The distance with respect to c_{out} between particles X_1 and X_2 is therefore calculated by subtracting the current setting of c_{out} with respect to particle X_2 from that of particle X_1 on a group-by-group basis to return Δc_{out} for each group.

5.2.6 Distance Calculation between Groups with respect to the Number of Blocks

Groups vary with respect to the number of ResNet blocks they contain. A strategy to temporarily pad the groups of both particles to the same length by adding empty blocks to the one with a fewer number of blocks is proposed, as shown in Fig. 34. The location of the empty block is used to decide where a block should be added or removed, as explained in Section 5.2.9.

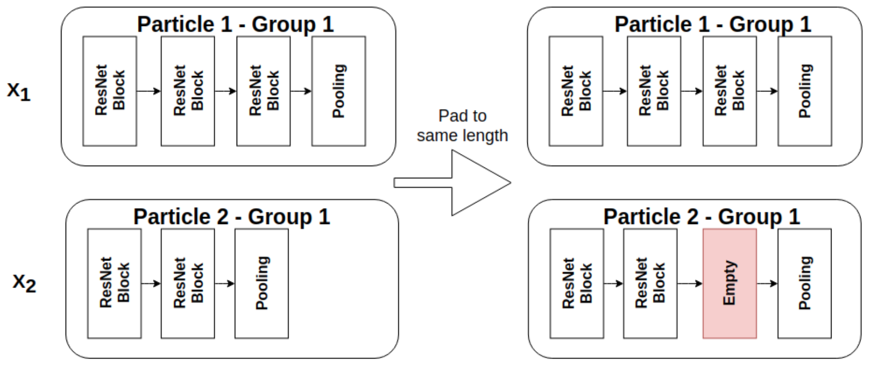


Figure 34: Groups from particles X_1 and X_2 which are temporarily padded to the same length in preparation for the particle distance calculation.

5.2.7 Distance Calculation with respect to the Block Kernel Size k

The distance between two blocks is computed with respect to the kernel size k by calculating $\Delta k = k_{(X_1 g_n b_m)} - k_{(X_2 g_n b_m)}$. Two special cases exist, i.e. (1) if the block from X_1 is an empty block as a result of the group padding described in Section 5.2.6, the output is also empty. Conversely, (2) if the block from X_2 is an empty block then the distance for the mth block is set as $\Delta k = k_{(X_1 g_n b_m)}$ as shown in Fig. 35.

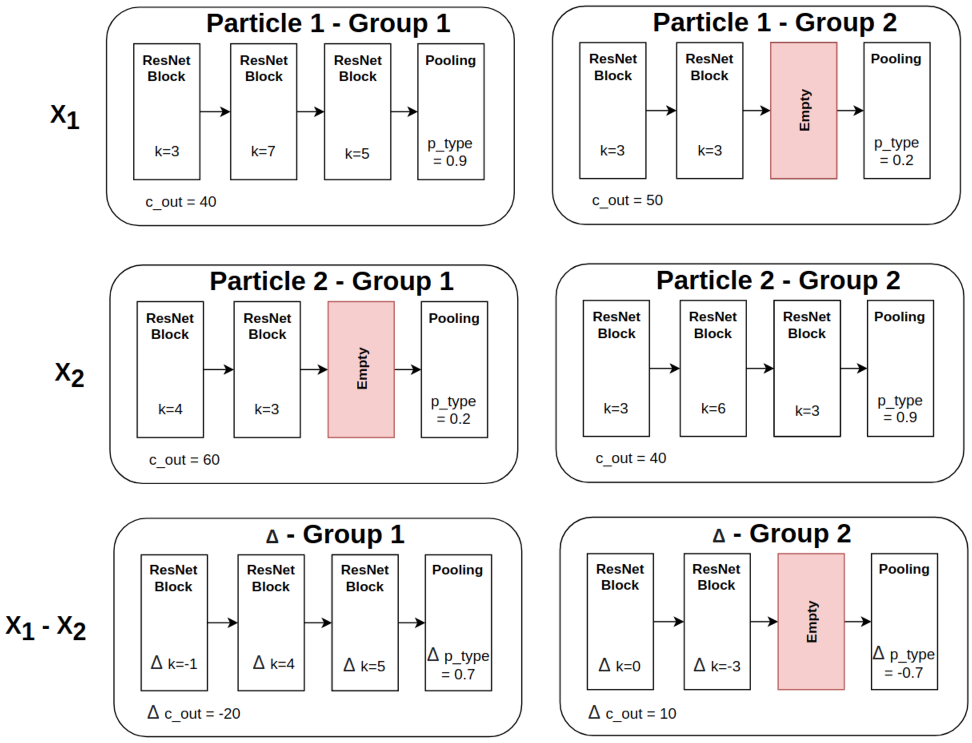


Figure 35: An example particle distance computation for $X_1 - X_2$

5.2.8 Distance Calculation with respect to the Pooling Type p_{type}

The distance between two pooling layers with respect to the kernel pooling type p_{type} is calculated by subtracting the pooling type from the respective group of X_2 from that of X_1, in order to calculate Δp_{type}.

5.2.9 Velocity Calculation

Within a PSO algorithm, the velocity V_i of particle i is calculated by computing the distances of particle position X_i with respect to the global best solution g_{best}, and its personal best solution p_{best}. A novel mechanism of velocity calculation is proposed with two new features, 1) calculates velocity based on g_{best} and a randomly selected neighbouring best solution, namely n_{best} and 2) allows velocity to be calculated based on the aforementioned encoding scheme. The hypothesis for using n_{best} instead of p_{best} within the proposed velocity calculation is to increase social communications and learning between neighbouring particles.

The adoption of randomly selected neighbouring elite solutions is able to add a degree of randomness and encourage global exploration before converging toward the swarm leader.

The neighbouring best position n_{best} is a non-uniform randomly selected particle from the swarm. Random selection is implemented using the python numpy [161] function random.choice(). The random.choice() function accepts a probability array containing values between 0 and 1. To compute the probability array, the losses of the entire swarm from the previous iteration $loss^{t-1}$ is used. Note that losses closer to 0 indicate better particle positions, and the intention is to favor such particles by giving them a higher probability. To achieve this, the losses are first inverted and then scaled to between 0 and 1. The resulting values are used to populate a probability array. A higher probability score for index i results a higher chance for particle i to be selected as n_{best}.

From the observations of the experiments, the selected neighbouring promising solution in each iteration is more likely to be one of the top solutions in the swarm. Such an operation not only adds randomness to the guiding signals in the search process but also ensures that the swarm is more likely to be guided towards optimal regions.

After determining n_{best}, the distance between n_{best} and the current particle i, i.e. $n_{best} - X_i$ is computed, as well as the distance between g_{best} and the particle i, i.e. $g_{best} - X_i$. Next, the network layers are iterated over. For every layer, a random number r is generated with a value between 0 and 1. The random number r is then compared against a threshold α. If $r \leq \alpha$, the distance of $g_{best} - X_i$ is selected, otherwise the distance of $n_{best} - X_i$ is adopted, as shown in Fig. 36. A similar process is also conducted for pooling layer generation.

The above velocity updating operation is conducted in three steps, i.e. 1) a selection is made for Δc_{out} for each group, 2) a selection is made for Δk, for each ResNet block within each group, and 3) a selection is made for Δp_{type} for each group. The velocity generation procedure is diversified by introducing randomness into the selection criteria in each step, as depicted in Equation 22. Owing to the guidance of multiple neighbouring and global elite solutions in velocity updating at block and group levels, the search process is equipped with better diversity and capabilities in overcoming local optima traps.

$$velocitySelection_{PerGroupAndBlock} = \begin{cases} g_{best} - X_i & \text{if } r \leq \alpha \\ n_{best} - X_i & \text{otherwise} \end{cases} \tag{22}$$

5.2.10 Position Updating

Once the new velocity V_i has been calculated, the position of particle X_i^{t+1} can be updated by adding the weighted velocity to the current particle position X_i^t. The weighting factor β controls the degree to which the new velocity is added to the current position. Higher values of β result in larger degrees of movement, whereby smaller values of β encourage a granular search of intermediate positions. The setting of $\beta = 0.5$ is selected for the experiences to give the best balance between exploration and exploitation. The modified position updating formula is provided in Equation 23.

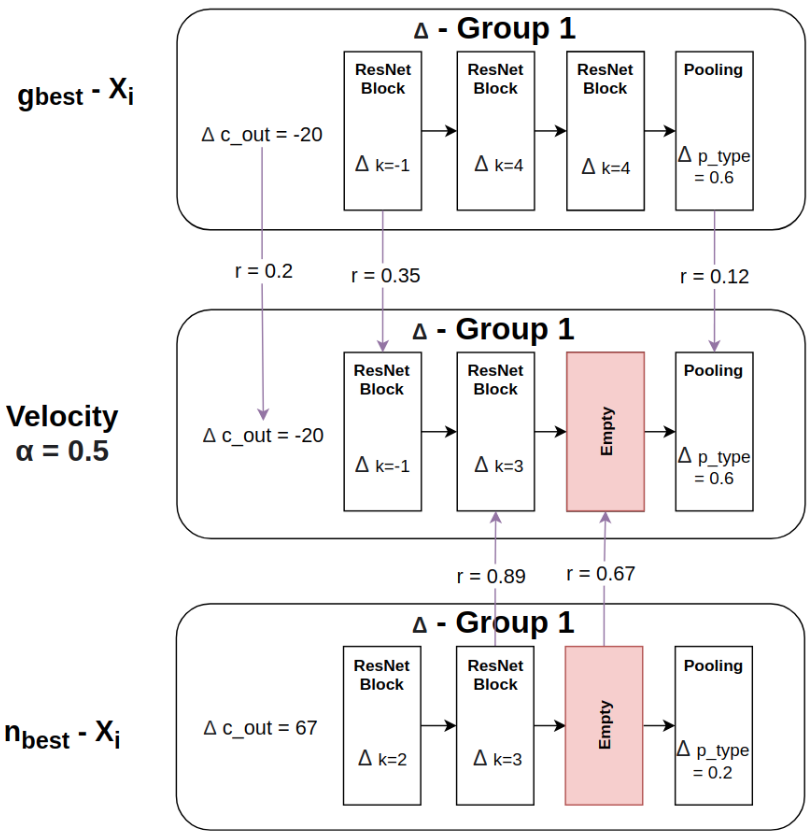


Figure 36: An example of velocity calculation between the selection of $n_{best} - X_i$ and $g_{best} - X_i$

Two special cases exist when applying velocity. (1) If an empty block within velocity V_i corresponds to a non-empty block in particle position X_i, the block is removed from the particle position X_i. (2) If a non-empty block within V_i corresponds to an empty block in particle position X_i, the block from velocity V_i is copied into particle position X_i. In this way, a positive value of k is ensured based on the definitions of both special cases.

$$X_i^{t+1} = \beta V_i + X_i^t \tag{23}$$

5.2.11 Fitness Evaluation

Each particle i within a swarm is evaluated by decoding the particle position X to construct a new CNN model. The new model is trained on a training set using the Adam optimizer [156] with a learning rate of 0.001 for 1 epoch. The average cross-entropy loss during training is used as the fitness score. The overall objective of the optimization process is to minimize the fitness scores by improving the particle positions over a number of iterations.

5.3 Experimental Studies

5.3.1 Data Sets

The proposed method is evaluated on six well-known benchmark data sets for direct comparison against closely related works so that the proposed method can be compared in similar settings. The test data sets are Rectangles-I, MNIST and four MNIST variant data sets. In comparison with the MNIST data set, the four MNIST variant data sets are more challenging owing to the transformations such as rotations, the addition of backgrounds, as well as other distracting factors.

Table 25 provides a summary of the aforementioned data sets including the official training and test split sample sizes.

Table 25: A summary of the data sets used in the experiments. All data sets have an input size of 28 x 28 x 1.

Data set	Description	Classes	Train/Test Samples
MNIST [1]	Handwritten digits	10	60,000/10,000
MNIST-RD [160] [162]	Rotated MNIST digits	10	12,000/50,000
MNIST-RB [160] [162]	MNIST digits with random background noise	10	12,000/50,000
MNIST-BI [160] [162]	MNIST digits with background images	10	12,000/50,000
MNIST-RD+BI [160] [162]	Rotated MNIST digits with background images	10	12,000/50,000
Rectangles-I [160] [162]	Rectangle border shapes with background images	2	12,000/50,000

5.3.2 Parameter Settings

The following settings shown in Table 26 are used in the experiments. Specifically, the maximum number of groups g_{max} is set as 2 owing to the selected input image sizes (28x28) from the data sets. As each group could potentially contain a pooling layer, and each pooling layer halves the output dimension size, adding more groups would negatively impact performance by reducing the output dimension too aggressively. For data sets containing larger input sizes, larger values for g_{max} could be selected.

The pooling layer type of a group is determined based on the current value of p_{type}, which has a value range between 0 and 1. The pooling operation selected based on the current value of p_{type} is explained in Equation 24.

Table 26: Algorithm settings and the search space used in the experiments. The settings are selected to closely match those of existing studies [10] so that a fair comparison can be made.

Name	Description	Value Used
k_{min}	Minimum kernel size	3
k_{max}	Maximum kernel size	7
out_{min}	Minimum number of channels	16
out_{max}	Maximum number of channels	256
b_{max}	Maximum number of blocks	15
g_{max}	Maximum number of groups	2
α	Layer selection boundary threshold	0.5
β	Velocity weighting factor	0.5

$$pooling = \begin{cases} NoPooling & \text{if } p_{type} \leq 0.33, \\ AvePooling & \text{if } p_{type} > 0.33 \ \& \ \ p_{type} \leq 0.66, \\ MaxPooling & \text{otherwise} \end{cases} \tag{24}$$

5.3.3 Benchmark Models

To test model efficiency, hand-crafted networks such as LeNet [1] and several deep architecture generation methods, i.e. IPPSO [13], MBO-ABCFE [139], GeNet [31], DNN-COCA [147], psoCNN [10] and sosCNN [14], are employed for performance comparison.

IPPSO [13] is a PSO-based approach for designing CNN architectures which employ a test methodology consisting of 20 particles over 10 iterations, constrained to a maximum of 9 convolutional layers with kernel sizes ranging between 1 and 8, and 3 fully connected layers.

MBO-ABCFE [139] is a variant of the MBO algorithm [138], which incorporates the search mechanisms of ABC [141] and FA [142] to increase exploration and exploitation of MBO, respectively. The experiment was conducted with a population size of 50, over 50 iterations and their optimized hyper parameters include the number of convolutional layers between 1 and 4, kernel size between 2 and 9, and a number of filters as either of the following, i.e. 8, 16, 32, 64, 128, 256, 512, and 1024. Their model also optimized the activation function type as either ReLU or a linear activation function, and batch size as 25, 50, 100 or 200.

GeNet [31] is a GA-based approach for deep network generation. GeNet adopts a block based architecture design with fixed kernel size and a fixed number of filters per block. Their work adopts a search strategy that optimizes the connections between network blocks, forming nonlinear block connection patterns. The model was evaluated using the MNIST data set achieving a 0.34% error rate, with a population size of 20 individuals over 50 iterations.

DNN-COCA [147] employs a multi-population strategy which adopts competitive and cooperative neuroevolution methods for the search of optimal architectures. The work was evaluated using the MNIST data set achieving an accuracy rate of 98.7% by evolving CNNs of depths between 5 and 7 layers.

psoCNN [10] is a PSO-based deep architecture generation strategy. It introduces a flexible encoding mechanism which describes a CNN model as an array. Each element within the array denotes a type of layer. The possible layer types comprise convolutions layers with kernel sizes between 3x3 and 7x7 inclusive and a maximum of 256 channels, as well as pooling layers and fully connected layers. Based on a selection criterion, the position of a particle within a swarm size of 20 is optimized over 10 iterations on a layer-by-layer basis. The new layer type and its settings are determined by choosing either the global or personal best solution.

Finally, sosCNN [14] is also selected for comparison. sosCNN is a SOS [119] based deep architecture generation method that generates CNNs without residual connections. This method uses a swarm of 20 individuals, optimized over 10 iterations. The hyperparameters selected for optimization are the layer types, i.e convolution layers with kernel sizes between 3x3 and 7x7, and up to 256 filters, pooling layers or fully connected layers. The maximum number of layers is limited to a total of 20.

5.3.4 Results

5.3.5 Performance Comparison with Existing Studies

The following settings in the experimental studies are used, i.e. a population of 20 and a maximum number of iterations of 10. During the search stage, each devised network gets trained with 1 epoch. The best model identified at the end of the search phase is trained for 100 epochs. Each experiment is repeated 10 times.

In Table 27, the results comparing the proposed approach against those of the aforementioned benchmark models, namely hand-crafted models, i.e. LeNet-1, LeNet-4 and LeNet-5 [1], as well as evolutionary based approaches, i.e. IPPSO [13], MBO-ABCFE [139], GeNet [31], DNN-COCA [147], psoCNN [10], and sosCNN [14] are reported. All reported results for the aforementioned benchmark models are taken from their respective publications for a fair comparison.

In the last two rows of Table 27, The best and mean classification error rates over 10 runs achieved by the proposed model are listed. The remaining rows are the best and the mean error rates (where available) reported by the compared methods in their original studies. The best performing results are highlighted in bold for a given data set.

As indicated by the reductions in the error rates reported across all the benchmark data sets, in comparison with the baseline methods, the proposed model shows better performances in most test cases. In addition, sosCNN is the best performing baseline method across all data sets. In Table 28, the error rates of the proposed model against those of

Table 27: Experimental results of the proposed method (resPsoCnn) and benchmark models, where the results of the benchmark models are extracted from their original studies.

Model	MNIST	MNIST-RD	MNIST-RB	MNIST-BI	MNIST-RD+BI	Rectangles-I
Hand-crafted architectures						
LeNet-1 [1]	1.70%	19.3%	7.50%	9.80%	40.06%	16.92%
LeNet-4 [1]	1.10%	11.79%	6.18%	8.96%	33.83%	16.09%
LeNet-5 [1]	0.95%	11.10%	5.99%	8.70%	34.64%	12.48%
Evolutionary algorithms for architecture generation						
IPPSO (best) [13]	1.13%	-	-	-	33%	-
IPPSO (mean) [13]	1.21%	-	-	-	34.50%	-
MBO-ABCFE (best) [139]	0.34%	-	-	-	-	-
GeNET (best) [31]	0.34%	-	-	-	-	-
DNN-COCA (mean) [147]	1.30%	-	-	-	-	-
psoCNN (best) [10]	0.32%	3.58%	1.79%	1.90%	14.28%	2.22%
psoCNN (mean) [10]	0.44%	6.42%	2.53%	2.40%	20.98%	3.94%
sosCNN (best) [14]	**0.30**%	3.01%	1.49%	**1.68**%	10.65%	1.57%
sosCNN (mean) [14]	0.40%	3.78%	1.89%	1.98%	13.61%	2.37%
resPsoCnn (best)	0.31%	**2.67**%	1.70%	1.74%	**8.76**%	1.19%
resPsoCnn (mean)	**0.33**%	**3.02**%	**1.76**%	**1.90**%	**9.27**%	**1.47**%

sosCNN [14] are listed, to clearly indicate performance improvements.

Table 28: The best and mean error rates over 10 runs for the proposed method (resPsoCnn) and sosCNN [14], where a (-) symbol indicates that the proposed model performs better whereas a (+) symbol indicates that the proposed model performs worse.

Model	MNIST	MNIST-RD	MNIST-RB	MNIST-BI	MNIST-RD+BI	Rectangles-I
resPsoCnn (best)	0.31%	**2.67%**	1.70%	1.74%	**8.76%**	**1.19%**
resPsoCnn (mean)	**0.33%**	**3.02%**	**1.76%**	**1.90%**	**9.27%**	**1.47%**
sosCNN (best) [14]	**0.30%**	3.01%	**1.49%**	**1.68%**	10.65%	1.57%
sosCNN (mean) [14]	0.40%	3.78%	1.89%	1.98%	13.61%	2.37%
error difference (best)	0.01%(+)	**-0.34%(-)**	0.21%(+)	0.06%(+)	**-1.89%(-)**	**-0.38%(-)**
error difference (mean)	**-0.07%(-)**	**-0.76%(-)**	**-0.13%(-)**	**-0.08%(-)**	**-4.34%(-)**	**-0.90%(-)**

MNIST represents a relatively simple handwritten digit classification problem with a small margin available for improvement. For MNIST, sosCNN is the best performing benchmark model, achieving a mean error rate of 0.40%. In comparison, the proposed approach achieves the highest mean error rate of 0.33%, which is an improvement of 0.07%.

For the MNIST-RD data set, sosCNN reported a mean error rate of 3.78%. The proposed model shows the lowest mean error rate of 3.02%, with an improvement of 0.76% over that of sosCNN. Moreover, the proposed model achieves a top 1 error rate of 2.67%, with an improvement of 0.34%, when compared with the top 1 result of sosCNN.

For the MNIST-RB data set, again the sosCNN method reported the lowest mean error rate of 1.89% among the benchmark models. The proposed approach obtains a mean error rate of 1.76%, with an improvement of 0.13%.

The proposed approach achieves a mean error rate of 1.90% with respect to the MNIST-BI data set,with an improvement of 0.08% over the mean error rate of 1.98% obtained by the best performing benchmark model sosCNN.

The MNIST-RD+BI data set depicts a more challenging classification problem due to consisting of rotated MNIST digits and background images. For this data set, sosCNN was the best performing benchmark model, reporting a mean error rate of 13.61%, and a top 1 error rate of 10.65%. The proposed approach shows a mean error rate of 9.27%, and a top 1 error rate of 8.76%, which indicate improvements of 4.34% and 1.89%, for the mean and best error rates, respectively.

For the Rectangles-I data set, sosCNN was again the best performing benchmark model, reporting a mean error rate of 2.37%, and a top 1 error rate of 1.57%. The proposed approach depicts a mean error rate of 1.47%, and a top 1 error rate of 1.19%, which indicate improvements of 0.38% and 0.90%, for the mean and best error rates, respectively.

5.3.6 Evaluation of the Proposed Encoding and Search Strategies

To identify the contributions to the overall results gained from the proposed encoding scheme and multiple leader guided search strategy, additional experiments have been conducted to isolate the two proposals. Rows 1 and 2 of Table 29 indicate the mean and top 1 error rates of the best performing benchmark model sosCNN. Rows 3 and 4 include the results of the encoding strategy in combination with the traditional PSO operation for position and velocity calculation. Rows 5 and 6 indicate the performance of the overall proposed model (i.e. the proposed encoding scheme in combination with the proposed search strategy, where the velocity calculation is based on the global best solution and a non-uniformly selected neighbouring best solution).

Table 29: Evaluation results of the state-of-the-art benchmark model sosCNN, the proposed encoding scheme in combination with the original PSO operation guided by the personal and global best solutions denoted as resPsoCnn-PB-GB, as well as the proposed encoding scheme in combination with the proposed search strategy guided by the neighbouring and global best solutions, denoted as resPsoCnn

Model	MNIST	MNIST-RD	MNIST-RB	MNIST-BI	MNIST-RD+BI	Rectangles-I
sosCNN (best) [14]	**0.30%**	3.01%	**1.49%**	**1.68%**	10.65%	1.57%
sosCNN (mean) [14]	0.40%	3.78%	1.89%	1.98%	13.61%	2.37%
resPsoCnn-PB-GB) (best)	**0.30%**	2.84%	1.51%	1.79%	9.20%	**0.89%**
resPsoCnn-PB-GB) (mean)	0.40%	3.23%	**1.76%**	2.02%	9.74%	1.66%
resPsoCnn (best)	0.31%(+)	**2.67%(-)**	1.70%(+)	1.74%(+)	**8.76%(-)**	1.19%(+)
resPsoCnn (mean)	**0.33%(-)**	**3.02%(-)**	**1.76%(-)**	**1.90%(-)**	**9.27%(-)**	**1.47%(-)**

For the MNIST data set, the mean and top 1 error rates of sosCNN are 0.40% and 0.30%, respectively. The proposed encoding strategy alone results in identical results. When the proposed encoding strategy is combined with the proposed search operation, the proposed method achieves a mean error rate of 0.33% which is an improvement of 0.07%, and a top 1 error rate of 0.31% within a reasonable margin of error.

For the MNIST-RD data set, the proposed encoding strategy in isolation achieves a mean error rate of 3.23%, and a top 1 error rate of 2.84%, with improvements of 0.55% and 0.17% over those of sosCNN, respectively. Furthermore, when the proposed encoding and search strategies are combined, further improvements are observed as indicated by a mean error rate improvement of 0.76% and a top 1 error rate improvement of 0.34%.

With respect to the MNIST-RB data set, the mean error rate reported by sosCNN is 1.89%. The mean error rates for the encoding strategy alone and the encoding strategy combined with the search strategy are both 1.76%, i.e an improvement of 0.13% against that of sosCNN. Furthermore, the results indicate that the encoding strategy in isolation performs the best with respect to the top 1 error rate than when combined with the proposed search strategy, with a top 1 error difference of 0.19% between these two versions of the proposed model.

For the MNIST-BI data set, the mean error rate reported by sosCNN is 1.98%. The mean error rate for the proposed encoding strategy alone is 2.02%, whilst when combined with the proposed search strategy, the error rate is improved to 1.90%, which is a 0.08% improvement against that of sosCNN. The best top 1 error rate, i.e. 1.68%, is achieved by sosCNN, whereas the proposed encoding scheme combined with the proposed search strategy achieves a top 1 error rate of 1.74%.

For the MNIST-RD+BI data set, the mean and the top 1 error rates using the proposed encoding strategy alone are 9.74% and 9.20%, respectively, with improvements of 3.87% and 1.45% over those of sosCNN, respectively. Furthermore, the mean and the top 1 error rates of the proposed encoding strategy combined with the proposed search strategy are 9.27%, with of 8.76%, respectively, with improvements of 4.34% and 1.89% over those of sosCNN.

For the Rectangles-I data set, the proposed encoding strategy in isolation achieves the lowest top 1 error rate of 0.89%, which is a 0.68% reduction when compared to the top 1 result of sosCNN. The proposed encoding strategy combined with the proposed search strategy achieves the lowest mean error rate of 1.47%, which is a 0.9% reduction than the mean result of sosCNN.

5.3.7 Theoretical Justification

In this research, firstly, a novel encoding scheme capable of describing deep CNN models comprising residual blocks has been proposed. Secondly, a new search strategy that updates the particle position based on the global best solution and a non-uniformly selected neighbouring best solution to overcome stagnation was proposed.

The version using the proposed encoding strategy isolated from the proposed search strategy is denoted as resPsoCnn-PB-GB. This method employs the original PSO operation guided by the personal and global best solutions found in related works such as psoCNN and sosCNN. The overall proposed model (i.e. the proposed encoding scheme combined with the proposed search strategy based on the global and neighbouring best solutions) is denoted as resPsoCnn, to demonstrate the benefits of the combination of both proposed strategies. The purpose of providing two sets of results is to demonstrate the contribution of each strategy in isolation from each other.

In Figure 37, the depths of the best networks produced by resPsoCnn-PB-GB and resPsoCnn against those of the benchmark models of IPPSO, psoCNN and sosCNN are compared. The empirical results indicate that resPsoCnn-PB-GB and resPsoCnn are capable of producing deeper network architectures as indicated by greater network depths across all data sets, owing to the introduction of residual connections. On the contrary, the related works show limited capabilities in producing deeper networks. Specifically, the related methods are limited in terms of the maximum depths due to vanishing gradients, which may impact a model's ability to learn. In the cases of vanishing gradients, backpropagation is unable to adjust weights as the gradients become 0, therefore the learning stops. Residual connections minimize vanishing gradient problems and therefore allow the two versions of the proposed model to form deeper networks. The empirical results in Table 27 indicate that the advantage of constructing deeper models is the improvement of performance, as indicated by the reductions in the mean error rates across all data sets for both resPsoCnn-PB-GB and resPsoCnn against those of the baseline methods.

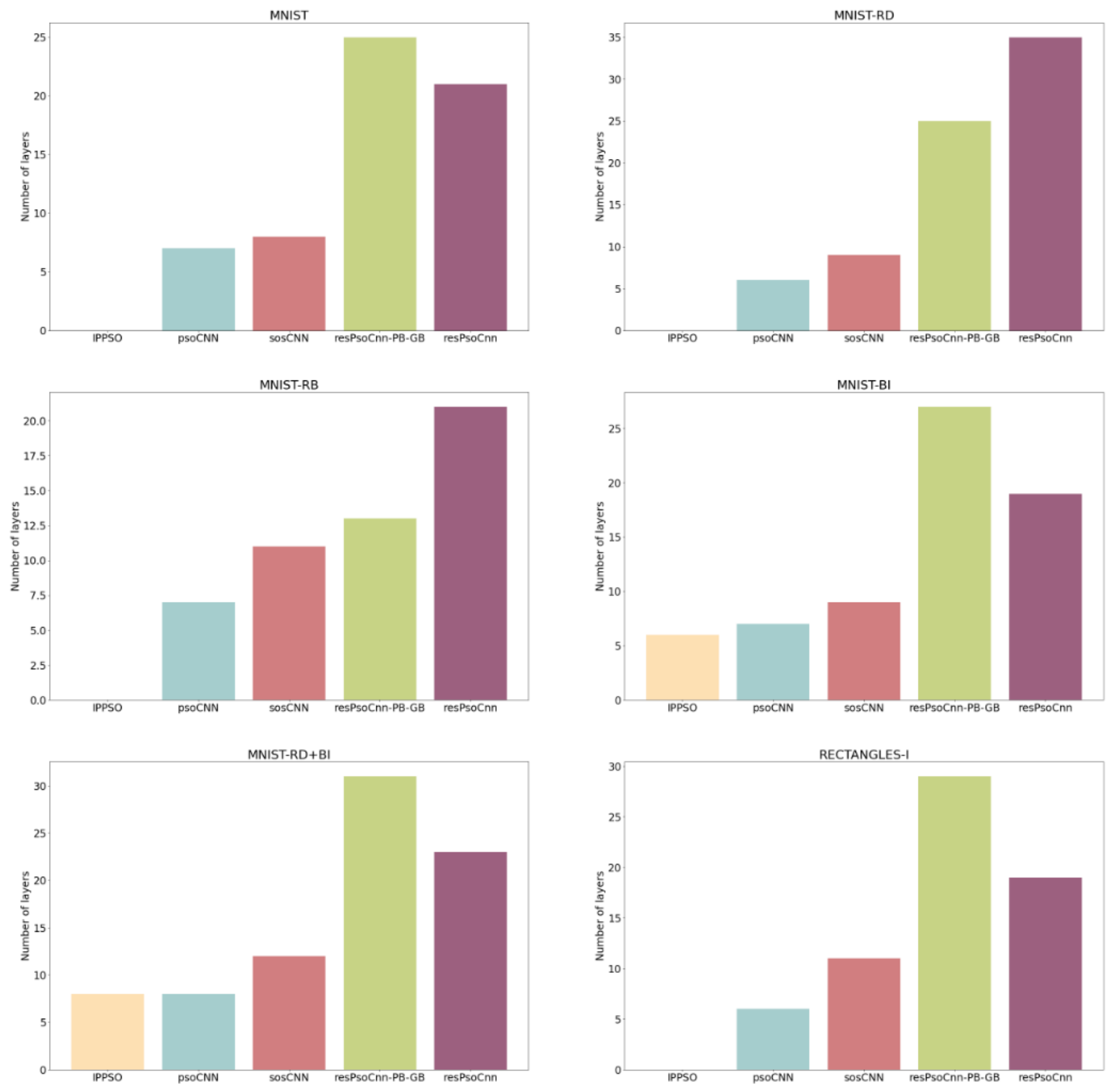


Figure 37: A comparison of model depths between those devised by the baseline benchmark models and the proposed approaches denoted as resPsoCnn-PB-GB and resPsoCnn, across all test data sets

Furthermore, the results in Table 29 indicate that resPsoCnn outperforms resPsoCnn-

PB-GB in terms of the mean error rates across all data sets, which confirms the benefits of combining the proposed encoding scheme and the search strategy based on the neighbouring and global best solutions. The empirical results indicate that the proposed search mechanism improves search diversity and avoids being trapped in a local optima owing to the randomness introduced by the non-uniform neighbouring selection strategy introduced in Section 5.2.9.

Table 30 presents the structures of the best models generated by resPsoCnn-PB-GB, while Table 31 presents the topologies of the best models devised by resPsoCnn. TB indicates a transitional block, and RB indicates a residual block as presented in Figure 33. FC denotes the final fully connected layer (i.e. the linear layer) of the model as presented in Figure 32. The models constructed by resPsoCnn tend to have more variety in the selected pooling layers. As indicated in Table 30, all pooling types selected by resPsoCnn-PB-GB are average pooling with the exception of MNIST-RD which selects no pooling for the first group. In contrast, as shown in Table 31, resPsoCnn selects max-pooling in the second groups for MNIST-RB and MNIST-BI, and no pooling for any groups for Rectangles-I. This greater variety suggests that resPsoCnn has more search diversity by selecting from the global and neighbouring best solutions for block and group configuration generation. It shows better capabilities in escaping from local optimum traps in relation to the pooling layer selection. Moreover the improvement of the search exploration has in turn resulted in the reductions in the error rates of resPsoCnn in comparison with those of resPsoCnn-PB-GB, as indicated in Table 29.

Table 30: The discovered best models for all benchmark data sets using resPsoCnn-PB-GB. TB indicates a transitional block which contains a single 1x1 convolutional layer and RB indicates a ResNet block which contains two convolutions, as indicated in Figure 33. FC indicates a fully connected layer.

Dataset	**Structure**
MNIST [1]	TB($c_{in} = 1$ $c_{out} = 177$) + RB(177x4x4) + RB(177x4x4) + RB(177x6x6) + AveragePool + TB($c_{in} = 177$ $c_{out} = 175$) + RB(175x6x6) + RB(175x6x6) + RB(175x5x5) + RB(175x3x3) + AveragePool + FC
MNIST-RD [160] [162]	TB($c_{in} = 1$ $c_{out} = 161$) + RB(161x5x5) + RB(161x7x7) + RB(161x6x6) + RB(161x6x6) + RB(161x4x4) + RB(161x5x5) + RB(161x7x7) + TB($c_{in} = 161$ $c_{out} = 115$) + RB(115x5x5) + RB(115x7x7) + RB(115x5x5) + RB(115x6x6) + RB(115x3x3) + RB(115x7x7) + RB(115x4x4) + AveragePool + FC
MNIST-RB [160] [162]	TB($c_{in} = 1$ $c_{out} = 153$) + RB(153x4x4) + RB(153x6x6) + + RB(153x4x4) + RB(153x3x3) AveragePool + TB($c_{in} = 153$ $c_{out} = 183$) + RB(183x4x4) + RB(183x6x6) + RB(183x7x7) + AveragePool + FC
MNIST-BI [160] [162]	TB($c_{in} = 1$ $c_{out} = 136$) + RB(136x4x4) + RB(136x3x3) + RB(136x5x5) + RB(136x3x3) + AveragePool + TB($c_{in} = 136$ $c_{out} = 136$) + RB(136x6x6) + RB(136x5x5) + RB(136x5x5) + RB(136x3x3) + RB(136x3x3) + RB(136x3x3) + AveragePool + FC
MNIST-RD+BI [160] [162]	TB($c_{in} = 1$ $c_{out} = 150$) + RB(231x5x5) + RB(231x5x5) + RB(231x7x7) + RB(231x3x3) + AveragePool + TB($c_{in} = 150$ $c_{out} = 98$) + RB(120x4x4) + RB(120x6x6) + RB(120x6x6) + RB(120x5x5) + AveragePool + FC
RECTANGLES-I [160] [162]	TB($c_{in} = 1$ $c_{out} = 195$) + RB(195x3x3) + RB(195x6x6) + RB(195x3x3) + AveragePool + TB($c_{in} = 195$ $c_{out} = 85$) + RB(85x7x7) + RB(85x5x5) + RB(85x3x3) + AveragePool + FC

Table 31: The discovered best models for all benchmark data sets using resPsoCnn. TB indicates a transitional block which contains a single 1x1 convolutional layer and RB indicates a ResNet block which contains two convolutions, as indicated in Figure 33. FC indicates a fully connected layer.

Dataset	Structure
MNIST [1]	TB($c_{in} = 1$ $c_{out} = 176$) + RB(176x4x4) + RB(176x5x5) + RB(176x5x5) + RB(176x4x4) + RB(176x5x5) + RB(176x3x3) + AveragePool + TB($c_{in} = 176$ $c_{out} = 198$) + RB(198x5x5) + RB(198x6x6) + RB(198x4x4) + RB(198x4x4) + AveragePool + FC
MNIST-RD [160] [162]	TB($c_{in} = 1$ $c_{out} = 184$) + RB(184x4x4) + RB(184x4x4) + RB(184x5x5) + RB(184x4x4) + RB(184x3x3) + RB(184x3x3) + AveragePool + TB($c_{in} = 184$ $c_{out} = 146$) + RB(146x4x4) + RB(146x4x4) + RB(146x5x5) + RB(146x3x3) + AveragePool + FC
MNIST-RB [160] [162]	TB($c_{in} = 1$ $c_{out} = 216$) + RB(216x5x5) + RB(216x6x6) + AveragePool + TB($c_{in} = 216$ $c_{out} = 158$) + RB(158x7x7) + RB(158x4x4) + MaxPool + FC
MNIST-BI [160] [162]	TB($c_{in} = 1$ $c_{out} = 188$) + RB(188x4x4) + RB(188x5x5) + RB(188x5x5) + RB(188x4x4) + RB(188x4x4) + RB(188x3x3) + RB(188x3x3) + AveragePool + TB($c_{in} = 188$ $c_{out} = 177$) + RB(177x5x5) + RB(177x3x3) + RB(177x3x3) + RB(177x3x3) + MaxPool + FC
MNIST-RD+BI [160] [162]	TB($c_{in} = 1$ $c_{out} = 231$) + RB(231x4x4) + RB(231x5x5) + RB(231x5x5) + RB(231x4x4) + RB(231x3x3) + RB(231x4x4) + AveragePool + TB($c_{in} = 231$ $c_{out} = 120$) + RB(120x5x5) + RB(120x5x5) + RB(120x3x3) + RB(120x4x4) + RB(120x4x4) + RB(120x3x3) + RB(120x3x3) + AveragePool + FC
RECTANGLES-I [160] [162]	TB($c_{in} = 1$ $c_{out} = 71$) + RB(71x4x4) + RB(71x3x3) + RB(71x7x7) + RB(71x6x6) + RB(71x6x6) + RB(71x6x6) + TB($c_{in} = 71$ $c_{out} = 21$) + RB(21x5x5) + RB(21x4x4) + RB(21x6x6) + RB(21x6x6) + RB(21x7x7) + RB(21x4x4) + FC

6 Conclusions

The main contributions of this research is the proposal of a new efficiency focused CNN model which achieves greater accuracy and more computational efficiency when compared to the current state-of-the-art efficiency focused CNN models. This work is important as resource constrained environments such as IoT deployments and CCTV cameras often lack GPU acceleration capabilities, therefore a more efficient and accurate model means more accurate models could run on the CPU. Furthermore, new strategies for automatically designing CNN models that embed human knowledge in a PSO based search strategies are introduced. The proposed PSO search strategies outperformed the current state-of-the-art approaches. Such an automated approach can be applied to other domains and data sets to generate custom models for different tasks. Automated approaches for model construction are important as they reduce the need to apply transfer learning to large base models unsuitable for resource constrained environments. Instead of transfer learning, automated approaches can be used to construct a new architecture specifically for the task at hand. Search bounds such as the maximum model depth and width can be set to ensure proposed solutions fit within the target hardware making such an approach suitable for all environments including resource constrained ones.

6.1 Efficiency Focused CNN Architecture Conclusion

The aim of the research in Section 3 was to propose an efficiency focused CNN model that reduces computational complexity whilst maximizing accuracy so that more accurate models can be deployed in constrained environments such as CCTV cameras. This raised the following research question:

- Is there a better way to construct a CNN model so that the model achieves greater accuracy levels with less computational cost, resulting in more accurate CNN models that can be deployed into resource contained environments such as IoT devices?

The research question was addressed through the development of IoTNet, a new efficiency focused CNN architecture that saves computation cost by factorising 3x3 standard convolutions into pairs of 1x3 and 3x1 standard convolutions, rather than performing depth-wise separable convolutions. Depth-wise separable convolutions used in existing efficiency focused CNN models to computational cost at the detriment of accuracy.

The experimental results of the proposed IoTNet model produced accuracy improvements and computational cost savings when compared against influential efficiency-focused architectures that save computational cost by utilising depth-wise separable convolutions. For example, IotNet outperformed MobileNetV2 on CIFAR-10 with an accuracy improvement of 13.43% with 39.63% fewer FLOPs, ShuffleNet on SVHN with an accuracy improvement of 6.49% with 31.84% fewer FLOPs and over MobileNet on GTSRB with an accuracy improvement of 5% with 0.38% fewer FLOPs.

The proposed model also outperformed the current state-of-the-art efficiency-focused model that incorporates 1x3 and 3x1 depth-wise separable convolutions to save computational cost such as EffNet (EffNet V1 or EffNet V2), across all data sets. IoTNet improved accuracy on CIFAR-10 by 9.7% with 13% fewer FLOPs, on SVHN by 1.92% with 58.5% fewer FLOPs and on GTSRB by 2.77% with 24.63% fewer FLOPs. The experimental studies also indicate that the proposed model delivers greater accuracy with a lower computational cost in comparison with those of the scaled-down 3x3 convolution-based counterpart model, representative of state-of-the-art WideResnet, ResNet and PyramidNet.

The main contributions of this research are as follows.

- A new efficiency focused CNN model is proposed, namely IoTNet. IoTNet was designed specifically for performance constrained environments such as IoT devices, CCTV or embedded systems. It trades accuracy with a reduction in computational cost differently from existing methods by employing novel pairs of 1x3 and 3x1 normal convolutions, rather than using depth-wise separable convolutions.
- An in-depth comparison of the proposed architecture against efficiency-focused models including MobileNet [5], MobileNetV2 [7], ShuffleNet [6] and EffNet [9] has been conducted using CIFAR-10 [18], Street View House Numbers (SVHN) [44] and German Traffic Sign Recognition Benchmark (GTSRB) [45] data sets. The empirical results indicate that the proposed block architecture constructed exclusively from pairs of 1x3 and 3x1 normal convolutions, with average pooling for downsampling, outperforms the current state-of-the-art depth-wise separable convolution-based architectures in terms of accuracy and cost.

The benefit of this work is that more powerful models can now be deployed within tightly constrained environments. which is significant as the use cases for CNNs within resource-constrained environments are extremely broad, e.g., IoT and smartphone-based deployments, medical diagnosis, image and video analysis. Lighter and faster models also enable researchers to prototype ideas faster with fewer resources. The empirical results indicate that the proposed architecture improves the trade-off between accuracy and computational cost over existing, depth-wise-based approaches which are typically used in efficiency-focused models.

6.2 Automating CNN Architecture Generation Conclusion

The aim of the research in Section 4 was to propose a PSO based search strategy for automatically designing efficient networks for the data set at hand, that produces more accurate models when compared against the current state-of-the-art approaches. This raised the following research question:

- Can CNN model architectures be encoded in a way that it is ensured to be both architecturally valid and reasonably built to avoid the need for additional hard coded governing rules [10] or wasteful function evaluations?

- Can each particle be guided effectively through a complex search space efficiently in order to construct more effective networks, that are faster than the current state-of-the-art algorithms [10] [11]?
- Can CNN models be designed to be more efficient using an algorithm that is both easy to understand and fast to run, so that the approach can be easily exploited in both academia and industry settings with limited specialised knowledge while not compromising the overall performance?

The first research question was achieved by proposing a group-based encoding strategy that removed the need for additional hard-coded rules. The proposed strategy adopts a natural particle movement and simplifies implementation.

The second research question was achieved by proposing a novel particle distance calculation scheme. It performs distance calculation at a parameter level to address the drawback of the existing methods which copy the layers directly at random from either p_{best} or g_{best}. Moreover, a weighted position updating mechanism has been developed, with the use of a weighting factor that provides granular movement control in the search space. Combining the proposed distance computation strategy with the new position updating mechanism, the proposed method is equipped with superior search diversity and is less dependent on good initialization, as compared with related methods such as psoCNN.

The third research question was achieved by performing an experimental analysis using identical settings to closely related studies, i.e. no data augmentation, the same swarm size, iteration numbers, and search ranges. The experimental results demonstrated that the proposed method achieves superior performance in eight benchmark data sets, and achieves up to 7.58% improvement in accuracy and up to 63% reduction in computational cost in comparison with those of existing methods. The convergence curves of the proposed model across all data sets indicate a steady reduction in loss, indicating convergence of the particles without being trapped in local optima.

The main contributions of this research are as follows.

- A new group-based encoding strategy was proposed. A group consists of an optional pooling layer for downsampling and at least one convolutional layer. The maximum number of groups is can be adjusted in accordance with the input image size. As the position of pooling layers is guaranteed within a group and the maximum frequency pooling occurs is set by the maximum number of groups, valid model architecture can be constructed without the need for additional governing rules.
- A new velocity updating mechanism is proposed that overcomes weaknesses in existing methods such as psoCNN [10]. The velocity calculation in psoCNN computes which layers to copy from the global or personal best position. PsoCNN is not able to explore intermediate layer settings. For example, if the current particle position for the

first block has a kernel size of 3, and the global best has a kernel size of 5, psoCNN cannot try using a kernel size of 4. The proposed velocity updating mechanism overcomes this limitation by identifying the differences between layer configurations like the kernel size and pooling types. The proposed approach is also less dependent on the requirement of a good random swarm initialization.

- A new position updating mechanism with weighted velocity strengths has been proposed. The granular position updating mechanism enables a thorough exploration of the search space and increases the likelihood of generating diversified network configurations. It employs a weighted strength of the velocity updates for new position generation, leading to the exploration of the search space in various forces and scales to increase the chances of formulating diversified networks. Such a granular movement also enables a better balance between intensification and diversification in order to increase the chances of finding global optimality.

- A comprehensive evaluation of the proposed model with several data sets is conducted. The proposed model compares favourably with the state-of-the-art models such as psoCNN [10] and notable alternative methods including EvoCNN [12]. Serving as a practical alternative to deep network generation, the proposed model achieves up to 7.58% improvement in accuracy and up to 63% reduction in computational cost, in comparison with those from the current state-of-the-art methods. Importantly, the proposed model is repeatable and easy to implement with limited hardware resources.

The benefit of the proposed approach is that without specialist knowledge of how to construct CNN architectures, the proposed approach can be used to automatically generate CNN models with greater accuracy than was possible using existing CNN automatic architecture generation techniques.

6.3 Automating Residual CNN Architecture Generation Conclusion

The aim of the research in Section 5 was to propose a PSO based search strategy capable of generating deeper CNN architectures owing to the inclusion of residual connections. This raised the following research question:

- Can a rich CNN model architecture search be performed whilst exploiting advanced techniques such as residual connections so that larger models can be constructed?

- How can a search be performed that is not susceptible to becoming trapped in local optima?

The first research question was achieved by proposing a novel residual based encoding strategy capable of overcoming the vanishing gradient problems so that larger models could be constructed. The results in Figure 37 indicate that the model is capable of generating deeper architectures than those yielded by closely related studies such as IPPSO, psoCNN and sosCNN, all of which do not exploit residual connections. In addition, the devised networks show better capabilities in tackling vanishing gradients owing to the adoption of residual connections. Moreover, the results also indicate that the deeper models produced by resPsoCnn-PB-GB lead to more accurate solutions, specifically an accuracy improvement on MNIST-RD, MNIST-RB, MNIST-RD+BI and Rectangles-I data sets, as indicated in Table 29.

The second research question was achieved by proposing a novel search process guided by neighbouring and global best solutions to avoid local optima traps. With respect to the proposed search strategy guided by neighbouring and global best solutions, the empirical results indicate that across all six data sets, that the proposed search strategy results in more accurate models when compared to the models yielded by closely related studies such as IPPSO, psoCNN and sosCNN, as indicated in Table 27.

In Table 29, the results of resPsoCnn show the effectiveness of the proposed search strategy as indicated by an improvement in model accuracy when compared directly against resPsoCnn-PB-GB, which adopts a traditional PSO search strategy, based on the global and personal best positions, i.e. identical to the strategies adopted by PSO, IPPSO, psoCNN and sosCNN.

Furthermore, the results in Table 27 indicate that the combination of the proposed movement and encoding strategies results in a further improvement of performance when compared to the related studies. Specifically, the proposed model resPsoCnn achieves the most significant improvement of 4.43% over the best baseline method, sosCNN, on MNIST-RD+BI.

The main contributions of this research are as follows.

- A novel PSO algorithm, namely resPsoCnn, is proposed for residual deep architecture generation. First, a new group-based encoding scheme is proposed which provides compatibility with residual connections. Each group contains one or more residual convolutional blocks and an optional pooling layer. Each residual block inside a group shares the same number of filters so that the residual operation can be performed. The number of filters per group is optimised to control the network width. Before each group, a transitional layer is used to increase or decrease the number of filters to the number required by the subsequent group. The kernel sizes of convolutional layers are individually encoded, giving fine-grained control over the receptive field of each block. The number of blocks within each group can vary in order to increase or decrease the model depth, while different pooling layer types are embedded to control downsampling.

- Proposing an optimization strategy that exploits the advantages of residual connections to avoid the vanishing gradient problem. Such a strategy addresses weaknesses in related studies which either 1) perform optimization tasks only on fixed skeleton models (e.g. fixed numbers of blocks) that make use of residual connections, but restrict diversity as settings such as kernel sizes and pooling types are fixed, or 2) do not use residual connections and instead optimize a range of hyperparameter settings, capable of producing diverse, but shallow networks. Our proposed strategy addresses both of the aforementioned weaknesses by providing the ability to make use of residual connections to construct deep network architectures, whilst also optimizing a range of network settings to improve diversity.

- Proposing a new velocity updating mechanism that adds randomness to the updating of the group and block hyperparameters. Specifically, it employs multiple elite signals, i.e. the swarm leader and the non-uniformly randomly selected neighbouring best solutions, for searching optimal hyper-parameters. The hyperparameter updating at the group and block levels is conducted by either selecting from the distance between the current particle and the global best solution, or the distance between the current particle and a neighbouring best solution, to increase search diversity. The proposed search mechanism optimizes the number of groups, network width and depth, kernel sizes, and pooling layer choices to produce a rich assortment of candidate best solutions of residual deep architectures. Owing to the guidance of multiple elite signals (e.g. the global best and neighbouring promising solutions), the proposed search process achieves a better balance between exploration and exploitation, and addresses a weakness in existing search methods where the search processes led by the single leader are prone to being trapped in local optima and converge prematurely. Evaluated using a number of benchmark data sets, the devised networks show superior performances against those yielded by several state-of-the-art existing methods.

The benefit of the proposed approach is that an automatic CNN architecture generation technique that considers residual connections got introduced. To the best of my knowledge, this is the first such work in this area. The results indicate that constructing residual CNN models produces deeper models, plus more accurate solutions. The benefits in respect to the neighbouring best search strategy could get exploited in other related areas. Other areas include all related works which adopt a traditional guided search that follows the global and personal best positions.

6.4 Limitations

Potential limitations around statistical significance tests when comparing against benchmark models exist. This study made direct comparisons with related works using the same methods for evaluation, specifically calculating the mean and max accuracy over 10 runs. Alternative tests such as Wilcoxon and rank sum do exist but the data to calculate such tests was not published by closely related studies. The proposed methods for automatically evolving convolutional neural networks contained a single object for the cost. Alternative cost functions considering multiple objectives such as minimizing the number of FLOPS, whilst maximizing accuracy has been considered for future work.

6.5 Future Direction

The proposed efficiency focused model IoTNet and proposed mechanisms for automatically generating deep CNN architectures has shown superior performance when compared against their respective state-of-the-art research. Despite this, future work could further enhance performance. The possible avenues for future research have been summaries in the following list:

- Augmentation strategies highlighted in related works such as [86] could be exploited to further improve performance. Methods such as these will be especially helpful on imbalanced data sets such as GTSRB.
- The proposed efficiency focused IoTNet model could be combined with the proposed CNN architecture generation techniques. Such work could consider using a cost function that calculates the cost based on the accuracy of a proposed CNN model and the computational cost for the proposed CNN model measured in floating point operations and memory footprint. Such work could further improve the proposed architecture.
- Research could be conducted to explore hybrid deep networks such as the combination of CNN with Long Short-Term Memory (LSTM) networks. Such hybrid models will be useful for undertaking various image understanding and time series prediction tasks.
- Different swarm sizes and iteration counts could be explored for the proposed CNN architecture generation techniques.
- The settings selected for CNN architecture optimization were based on the constraints reported in the related paper. [10], in order to ensure that a fair comparison can be made. It is envisaged that optimal settings can be formulated, e.g. with the use of adaptive strategies for α and β. Future work could benefit from using an adaptive strategy to alter α slightly after each iteration so that the position differences to the global best solution are given more emphasis than those from earlier iterations.
- Hybrid search strategies based on the integration of the proposed CNN architecture generation techniques with other swarm intelligence algorithms, e.g. Firefly Algorithm [142], Cuckoo Search [109], Dragonfly Algorithm [163] and Grey Wolf Optimizer [164] could further enhance search capabilities.
- Applying CNN architecture generation techniques to other domains such as object detection, semantic segmentation, video description and visual question generation may result in the discovery of novel object detection models. Such work could also consider exploring the proposed 1x3 and 3x1 kernel sizes when considering computational cost reduction.

- When performing CNN architecture generation, future research could be conducted that considers adding dense connections and optimizing the connection types. For example, the optimization process could be adapted to select between residual and dense connections. As a more general form, each CNN layer could be encoded with metadata to define connections with zero or many other layers.

- Related works in the field of automated data augmentation discovery could be combined with the proposed CNN architecture generation techniques.

- An exponential moving average of the most recent velocities used to update particle positions could be considered. Such an approach could smooth particle movements.

6.6 Application of Research Over 3-5 Years

This research resulted in the proposal of a new efficiency focused CNN model architecture. The proposed efficiency focused CNN model save computational cost by factorising 3x3 standard convolutions into pairs of 1x3 and 3x1 standard convolutions. This contribution has advanced existing knowledge by proposing a novel CNN model construction that better trades off accuracy with computation cost when compared to existing research methods. The applications for this advancement over the next 3-5 years benefit both academia and industry. Efficiency focused models are important as they require less expensive hardware to train, making training such models faster and accessible. Efficient CNN models require fewer resources to run, making it possible to run CNN models on resource-constrained devices such as CCTV cameras and IoT devices with greater accuracy than what was possible before. Since the publication of the proposed model, researchers at Google have also published work containing CNN models constructed from 1x3 and 3x1 kernels [165] that focused on training keyword spotters with limited training data. Such publications show that the applications over the next 3-5 years covers many research topics. In industry, a model which can achieve greater accuracy with less computational resources means that such a model is easier to scale, and are more suitable for edge deployments in limited hardware resources. In academia, a model achieving greater accuracy with fewer resource requirements means that more novel research can be conducted quicker and in constrained environments such as robotics research and wearable devices. Furthermore, researchers with limited hardware availability can also use such methods to conduct research.

The research conducted that proposed new methods for automatically constructing CNN models have advanced existing knowledge by proposing methods for automatically constructing more accurate CNN models. The proposed methods introduced novelties including new search and encoding strategies. Such advances have applications in both academia and industry. Over the next 3-5 years, the proposed approaches for automatically generating CNN models means that new and novel problems can be solved without the need for in depth knowledge in CNN model construction. For research settings, the novelty can come from the application area rather than the model construction. In such research given a new and novel data set, the proposed approach could be applied to construct a suitable model to solve the task at hand. For example, [166] referenced the proposed approach when conducting a PSO based approach for automatic CNN architecture generation within the domain of tuberculosis X-Ray image classification. Within similar research areas, the advancement in knowledge could be built upon over the next 3-5 years. Likewise in industry, rapid prototyping and development could be conducted with less manual trial and error.

References

[1] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[4] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[5] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[6] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018.

[7] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[8] Ziyang He, Xiaoqing Zhang, Yangjie Cao, Zhi Liu, Bo Zhang, and Xiaoyan Wang. Litenet: Lightweight neural network for detecting arrhythmias at resource-constrained mobile devices. *Sensors*, 18(4):1229, 2018.

[9] Ido Freeman, Lutz Roese-Koerner, and Anton Kummert. Effnet: An efficient structure for convolutional neural networks. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 6–10. IEEE, 2018.

[10] Francisco Erivaldo Fernandes Junior and Gary G Yen. Particle swarm optimization of deep neural networks architectures for image classification. *Swarm and Evolutionary Computation*, 49:62–74, 2019.

[11] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

[12] Yanan Sun, Bing Xue, Mengjie Zhang, and Gary G Yen. Evolving deep convolutional neural networks for image classification. *IEEE Transactions on Evolutionary Computation*, 24(2):394–407, 2019.

[13] Bin Wang, Yanan Sun, Bing Xue, and Mengjie Zhang. Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2018.

[14] Fahui Miao, Li Yao, and Xiaojie Zhao. Evolving convolutional neural networks by symbiotic organisms search algorithm for image classification. *Applied Soft Computing*, 109:107537, 2021.

[15] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

[16] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[17] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[18] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[19] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In *International conference on artificial neural networks*, pages 270–279. Springer, 2018.

[20] Mengjun Zeng and Nanfeng Xiao. Effective combination of densenet and bilstm for keyword spotting. *IEEE Access*, 7:10767–10775, 2019.

[21] Swarnambiga Ayyachamy, Varghese Alex, Mahendra Khened, and Ganapathy Krishnamurthi. Medical image retrieval using Resnet-18. In Po-Hao Chen and Peter R. Bak, editors, *Medical Imaging 2019: Imaging Informatics for Healthcare, Research, and Applications*, volume 10954, pages 233 – 241. International Society for Optics and Photonics, SPIE, 2019.

[22] Guangfen Wei, Gang Li, Jie Zhao, and Aixiang He. Development of a lenet-5 gas identification cnn structure for electronic noses. *Sensors*, 19(1):217, 2019.

[23] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[24] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *International conference on artificial neural networks*, pages 92–101. Springer, 2010.

[25] Rich Caruana, Steve Lawrence, and C Lee Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Advances in neural information processing systems*, pages 402–408, 2001.

[26] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

[27] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. Understanding the effective receptive field in deep convolutional neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 4905–4913, 2016.

[28] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48, 2019.

[29] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

[30] Edmar Rezende, Guilherme Ruppert, Tiago Carvalho, Fabio Ramos, and Paulo De Geus. Malicious software classification using transfer learning of resnet-50 deep neural network. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1011–1014. IEEE, 2017.

[31] Lingxi Xie and Alan Yuille. Genetic cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1388–1397, 2017.

[32] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE, 1995.

[33] Y. Sun, G. G. Yen, and Z. Yi. Evolving unsupervised deep neural networks for learning meaningful representations. *IEEE Transactions on Evolutionary Computation*, 23(1):89–103, 2019.

[34] Xin-She Yang. Nature-inspired optimization algorithms: Challenges and open problems. *Journal of Computational Science*, 46:101104, 2020. 20 years of computational science.

[35] Bin Wang, Yanan Sun, Bing Xue, and Mengjie Zhang. A hybrid differential evolution approach to designing deep convolutional neural networks for image classification. In

Tanja Mitrovic, Bing Xue, and Xiaodong Li, editors, *AI 2018: Advances in Artificial Intelligence*, pages 237–250, Cham, 2018. Springer International Publishing.

[36] John Henry Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[37] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[38] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[39] Ben Fielding and Li Zhang. Evolving deep denseblock architecture ensembles for image classification. *Electronics*, 9(11), 2020.

[40] B. Wang, B. Xue, and M. Zhang. Particle swarm optimisation for evolving deep neural networks for image classification by evolving and stacking transferable blocks. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, 2020.

[41] Kuruge Abeyrathna and Chawalit Jeenanunta. Escape local minima with improved particle swarm optimization algorithm. 11 2019.

[42] D. Steinkraus, I. Buck, and P.Y. Simard. Using gpus for machine learning algorithms. In *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, pages 1115–1120 Vol. 2, 2005.

[43] Morty Eisen. Introduction to poe and the ieee802. 3af and 802.3 at standards. *Marcum Technology: New York, NY, USA*, 2009.

[44] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Ng. Reading digits in natural images with unsupervised feature learning. *NIPS*, 01 2011.

[45] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In *IEEE International Joint Conference on Neural Networks*, pages 1453–1460, 2011.

[46] Tom Lawrence and Li Zhang. Iotnet: An efficient and accurate convolutional neural network for iot devices. *Sensors*, 19(24), 2019.

[47] Tom Lawrence, Li Zhang, Chee Peng Lim, and Emma-Jane Phillips. Particle swarm optimization for automatically evolving convolutional neural networks for image classification. *IEEE Access*, 9:14369–14386, 2021.

[48] Tom Lawrence, Li Zhang, Kay Rogage, and Chee Peng Lim. Evolving deep architecture generation with residual connections for image classification using particle swarm optimization. *Sensors*, 21(23), 2021.

[49] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, 1992.

[50] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[51] John S Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*, pages 227–236. Springer, 1990.

[52] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[53] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[54] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[55] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[56] Dongyoon Han, Jiwhan Kim, and Junmo Kim. Deep pyramidal residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5927–5935, 2017.

[57] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[58] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[59] Rahul Mazumder, Jerome H Friedman, and Trevor Hastie. Sparsenet: Coordinate descent with nonconvex penalties. *Journal of the American Statistical Association*, 106(495):1125–1138, 2011.

[60] Fatma Karray, Mohamed W Jmal, Alberto Garcia-Ortiz, Mohamed Abid, and Abdulfattah M Obeid. A comprehensive survey on wireless sensor node hardware platforms. *Computer Networks*, 144:89–110, 2018.

[61] Roberto Saia, Salvatore Carta, Diego Reforgiato Recupero, and Gianni Fenu. Internet of entities (ioe): A blockchain-based distributed paradigm for data exchange between wireless-based devices. In *8th International Conference on Sensor Networks, SENSORNETS 2019*, pages 77–84. SciTePress, 2019.

[62] Fernando Castaño, Stanisław Strzelczak, Alberto Villalonga, Rodolfo E Haber, and Joanna Kossakowska. Sensor reliability in cyber-physical systems using internet-of-things data: A review and case study. *Remote Sensing*, 11(19):2252, 2019.

[63] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[64] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[65] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[66] George B Moody and Roger G Mark. The impact of the mit-bih arrhythmia database. *IEEE Engineering in Medicine and Biology Magazine*, 20(3):45–50, 2001.

[67] A. S. Gaikwad and M. El-Sharkawy. Pruning convolution neural network (squeezenet) using taylor expansion-based criterion. In *2018 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*, pages 1–5, Dec 2018.

[68] P. Singh, V. S. R. Kadi, N. Verma, and V. P. Namboodiri. Stability based filter pruning for accelerating deep cnns. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1166–1174, Jan 2019.

[69] Mikel Galar, Alberto Fernández, Edurne Barrenechea, Humberto Bustince, and Francisco Herrera. Ordering-based pruning for improving the performance of ensembles of classifiers in the framework of imbalanced datasets. *Information Sciences*, 354:178–196, 2016.

[70] S. Lin, R. Ji, Y. Li, C. Deng, and X. Li. Toward compact convnets via structure-sparsity regularized filter pruning. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–15, 2019.

[71] Yann LeCun et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, 20:5, 2015.

[72] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[73] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[74] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[75] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2018.

[76] Z. Zhong, J. Yan, W. Wu, J. Shao, and C. Liu. Practical block-wise neural network architecture generation. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2423–2432, June 2018.

[77] E. Bochinski, T. Senst, and T. Sikora. Hyper-parameter optimization for convolutional neural network committees based on evolutionary algorithms. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3924–3928, Sep. 2017.

[78] Yanan Sun, Bing Xue, Mengjie Zhang, and Gary G Yen. Automatically designing cnn architectures using genetic algorithm for image classification. *arXiv preprint arXiv:1808.03818*, 2018.

[79] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[80] Michel Tokic. Adaptive ε-greedy exploration in reinforcement learning based on value differences. In *Annual Conference on Artificial Intelligence*, pages 203–210. Springer, 2010.

[81] David E Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of genetic algorithms*, volume 1, pages 69–93. Elsevier, 1991.

[82] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 722–737, 2018.

[83] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees 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*, pages 65–74. ACM, 2017.

[84] Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. Training deep neural networks with 8-bit floating point numbers. In *Advances in neural information processing systems*, pages 7675–7684, 2018.

[85] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[86] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 113–123, 2019.

[87] Treesukon Treebupachatsakul and Suvit Poomrittigul. Bacteria classification using image processing and deep learning. In *2019 34th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC)*, pages 1–3, 2019.

[88] Shoji Kido, Yasusi Hirano, and Noriaki Hashimoto. Detection and classification of lung abnormalities by use of convolutional neural network (cnn) and regions with cnn features (r-cnn). In *2018 International Workshop on Advanced Image Technology (IWAIT)*, pages 1–4, 2018.

[89] Guoming Chen, Yongchang Chen, Zeduo Yuan, Xuming Lu, Xiongyong Zhu, and Wanyi Li. Breast cancer image classification based on cnn and bit-plane slicing. In *2019 International Conference on Medical Imaging Physics and Engineering (ICMIPE)*, pages 1–4, 2019.

[90] Zhang Jin, Luo Qingli, Li Yu, Feng Hao, and Wei Jujie. Oil spill detection using refined convolutional neural network based on quad-polarimetric sar images. In *2019 14th IEEE International Conference on Electronic Measurement Instruments (ICEMI)*, pages 528–536, 2019.

[91] Kan Zeng and Yixiao Wang. A deep convolutional neural network for oil spill detection from spaceborne sar images. *Remote Sensing*, 12(6):1015, 2020.

[92] Samir S Yadav and Shivajirao M Jadhav. Deep convolutional neural network based medical image classification for disease diagnosis. *Journal of Big Data*, 6(1):1–18, 2019.

[93] Mengmeng Zhang, Wei Li, and Qian Du. Diverse region-based cnn for hyperspectral image classification. *IEEE Transactions on Image Processing*, 27(6):2623–2634, 2018.

[94] Chih-Cheng Chen, Zhen Liu, Guangsong Yang, Chia-Chun Wu, and Qiubo Ye. An improved fault diagnosis using 1d-convolutional neural network model. *Electronics*, 10(1):59, 2021.

[95] Benjamin Staar, Michael Lütjen, and Michael Freitag. Anomaly detection with convolutional neural networks for industrial surface inspection. *Procedia CIRP*, 79:484 – 489, 2019. 12th CIRP Conference on Intelligent Computation in Manufacturing Engineering, 18-20 July 2018, Gulf of Naples, Italy.

[96] T. Y. Tan, L. Zhang, C. P. Lim, B. Fielding, Y. Yu, and E. Anderson. Evolving ensemble models for image segmentation using enhanced particle swarm optimization. *IEEE Access*, 7:34004–34019, 2019.

[97] X. Zhou, L. Qian, P. You, Z. Ding, and Y. Han. Fall detection using convolutional neural network with multi-sensor fusion. In *2018 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*, pages 1–5, 2018.

[98] Philip Kinghorn, Li Zhang, and Ling Shao. A region-based image caption generator with refined descriptions. *Neurocomputing*, 272:416 – 424, 2018.

[99] Philip Kinghorn and Li Zhang. hierarchical and regional deep learning architecture for image description generation. *Pattern Recognition Letters*, 119:77–85, 2019.

[100] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[101] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[102] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[103] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(2):386–397, 2020.

[104] Xin-She Yang. Firefly algorithms for multimodal optimization. In *International symposium on stochastic algorithms*, pages 169–178. Springer, 2009.

[105] Taha Mostafaie, Farzin Modarres Khiyabani, and Nima Jafari Navimipour. A systematic study on meta-heuristic approaches for solving the graph coloring problem. *Computers and Operations Research*, 120:104850, 2020.

[106] Chunfeng Wang and Kui Liu. A randomly guided firefly algorithm based on elitist strategy and its applications. *IEEE Access*, 7:130373–130387, 2019.

[107] John H Holland. Genetic algorithms. *Scientific american*, 267(1):66–73, 1992.

[108] M. Dorigo and L.M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[109] X. Yang and Suash Deb. Cuckoo search via lévy flights. In *2009 World Congress on Nature Biologically Inspired Computing (NaBIC)*, pages 210–214, 2009.

[110] Micheal F Shlesinger, George M Zaslavsky, and Uriel Frisch. Lévy flights and related topics in physics. 1995.

[111] Jinjin Ding, Qunjin Wang, Qian Zhang, Qiubo Ye, and Yuan Ma. A hybrid particle swarm optimization-cuckoo search algorithm and its engineering applications. *Mathematical Problems in Engineering*, 2019, 2019.

[112] Li Huang, Shuai Ding, Shouhao Yu, Juan Wang, and Ke Lu. Chaos-enhanced cuckoo search optimization algorithms for global optimization. *Applied Mathematical Modelling*, 40(5):3860–3875, 2016.

[113] Kamlesh Mistry, Li Zhang, Siew Chin Neoh, Chee Peng Lim, and Ben Fielding. A micro-ga embedded pso feature selection approach to intelligent facial emotion recognition. *IEEE transactions on cybernetics*, 47(6):1496–1509, 2016.

[114] Teck Yan Tan, Li Zhang, and Chee Peng Lim. Intelligent skin cancer diagnosis using improved particle swarm optimization and deep learning models. *Applied Soft Computing*, 84:105725, 2019.

[115] Teck Yan Tan and Li Zhang. Adaptive melanoma diagnosis using evolving clustering, ensemble and deep neural networks. *Knowledge-Based Systems*, 187:104807, 2020.

[116] Worawut Srisukkham, Li Zhang, Siew Chin Neoh, Stephen Todryk, and Chee Peng Lim. Intelligent leukaemia diagnosis with bare-bones pso based feature optimization. *Applied soft computing*, 56:405–419, 2017.

[117] Ben Fielding and Li Zhang. Evolving image classification architectures with enhanced particle swarm optimisation. *IEEE Access*, 6:68560–68575, 2018.

[118] Ben Fielding, Tom Lawrence, and Li Zhang. Evolving and ensembling deep cnn architectures for image classification. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.

[119] Min-Yuan Cheng and Doddy Prayogo. Symbiotic organisms search: A new metaheuristic optimization algorithm. *Computers and Structures*, 139:98–112, 2014.

[120] Bin Wang, Yanan Sun, Bing Xue, and Mengjie Zhang. Evolving deep neural networks by multi-objective particle swarm optimization for image classification. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, page 490–498, New York, NY, USA, 2019. Association for Computing Machinery.

[121] Tulika Dutta, Sandip Dey, Siddhartha Bhattacharyya, and Somnath Mukhopadhyay. Quantum fractional order darwinian particle swarm optimization for hyperspectral multi-level image thresholding. *Applied Soft Computing*, page 107976, 2021.

[122] Daniela Szwarcman, Daniel Civitarese, and Marley Vellasco. Quantum-inspired neural architecture search. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2019.

[123] Li Zhang, Chee Peng Lim, and Yonghong Yu. Intelligent human action recognition using an ensemble model of evolving deep networks with swarm-based optimization. *Knowledge-Based Systems*, 220, 2021.

[124] Xiaobo Liu, Chaochao Zhang, Zhihua Cai, Jianfeng Yang, Zhilang Zhou, and Xin Gong. Continuous particle swarm optimization-based deep learning architecture search for hyperspectral image classification. *Remote Sensing*, 13(6), 2021.

[125] Chia-Feng Juang, Yu-Cheng Chang, and I-Fang Chung. Optimization of recurrent neural networks using evolutionary group-based particle swarm optimization for hexapod robot gait generation. *Hybrid Metaheuristics: Research And Applications*, 84:227, 2018.

[126] Teck Yan Tan, Li Zhang, and Chee Peng Lim. Intelligent skin cancer diagnosis using improved particle swarm optimization and deep learning models. *Applied Soft Computing*, 84:105725, 2019.

[127] Li Zhang and Chee Peng Lim. Intelligent optic disc segmentation using improved particle swarm optimization and evolving ensemble models. *Applied Soft Computing*, 92:106328, 2020.

[128] Teck Yan Tan, Li Zhang, and Chee Peng Lim. Adaptive melanoma diagnosis using evolving clustering, ensemble and deep neural networks. *Knowledge-Based Systems*, 187:104807, 2020.

[129] Long Zhang and Lin Zhao. High-quality face image generation using particle swarm optimization-based generative adversarial networks. *Future Generation Computer Systems*, 122:98–104, 2021.

[130] Zhongke Gao, Yanli Li, Yuxuan Yang, Xinmin Wang, Na Dong, and Hsiao-Dong Chiang. A gpso-optimized convolutional neural networks for eeg-based emotion recognition. *Neurocomputing*, 380:225 – 235, 2020.

[131] Y. Sun, B. Xue, M. Zhang, and G. G. Yen. Completely automated cnn architecture design based on blocks. *IEEE Transactions on Neural Networks and Learning Systems*, 31(4):1242–1254, 2020.

[132] Yanan Sun, Bing Xue, Mengjie Zhang, Gary G. Yen, and Jiancheng Lv. Automatically designing cnn architectures using the genetic algorithm for image classification. *IEEE Transactions on Cybernetics*, 50(9):3840–3854, 2020.

[133] Brad L Miller, David E Goldberg, et al. Genetic algorithms, tournament selection, and the effects of noise. *Complex systems*, 9(3):193–212, 1995.

[134] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8697–8710, 2018.

[135] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, pages 1729–1738, 2019.

[136] A. Kwasigroch, M. Grochowski, and A. Mikołajczyk. Neural architecture search for skin lesion classification. *IEEE Access*, 8:9061–9071, 2020.

[137] Edvinas Byla and Wei Pang. "deepswarm: Optimising convolutional neural networks using swarm intelligence". In Zhaojie Ju, Longzhi Yang, Chenguang Yang, Alexander Gegov, and Dalin Zhou, editors, *Advances in Computational Intelligence Systems. UKCI 2019.*, Advances in Intelligent Systems and Computing, pages 119–130. Springer, August 2019.

[138] Gai-Ge Wang, Suash Deb, and Zhihua Cui. Monarch butterfly optimization. *Neural computing and applications*, 31(7):1995–2014, 2019.

[139] Nebojsa Bacanin, Timea Bezdan, Eva Tuba, Ivana Strumberger, and Milan Tuba. Monarch butterfly optimization based convolutional neural network design. *Mathematics*, 8(6), 2020.

[140] Ivana Strumberger, Eva Tuba, Nebojsa Bacanin, Marko Beko, and Milan Tuba. Modified and hybridized monarch butterfly algorithms for multi-objective optimization. In Ana Maria Madureira, Ajith Abraham, Niketa Gandhi, and Maria Leonilde Varela, editors, *Hybrid Intelligent Systems*, pages 449–458, Cham, 2020. Springer International Publishing.

[141] Nebojsa Bacanin and Milan Tuba. Artificial bee colony (abc) algorithm for constrained optimization improved with genetic operators. *Studies in Informatics and Control*, 21:137–146, 06 2012.

[142] XS Yang. Firefly algorithm, nature inspired metaheuristic algorithms, 2010, 2010.

[143] Dechao Chen, Xiang Li, and Shuai Li. A novel convolutional neural network model based on beetle antennae search optimization algorithm for computerized tomography diagnosis. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–12, 2021.

[144] Jiangyu Wang and Huanxin Chen. Bsas: Beetle swarm antennae search algorithm for optimization problems. *arXiv preprint arXiv:1807.10470*, 2018.

[145] Ching-Hung Lee, Wei-Yu Lai, and Yu-Ching Lin. A tsk-type fuzzy neural network (tfnn) systems for dynamic systems identification. In *42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475)*, volume 4, pages 4002–4007. IEEE, 2003.

[146] Meng Li, William Hsu, Xiaodong Xie, Jason Cong, and Wen Gao. Sacnn: Self-attention convolutional neural network for low-dose ct denoising with self-supervised perceptual loss network. *IEEE transactions on medical imaging*, 39(7):2289–2301, 2020.

[147] Sreenivas Sremath Tirumala. Evolving deep neural networks using coevolutionary algorithms with multi-population strategy. *Neural Computing and Applications*, 32(16):13051–13064, 2020.

[148] Maria G. Baldeon Calisto and Susana K. Lai-Yuen. Self-adaptive 2D-3D ensemble of fully convolutional networks for medical image segmentation. In Ivana Išgum and Bennett A. Landman, editors, *Medical Imaging 2020: Image Processing*, volume 11313, pages 459 – 469. International Society for Optics and Photonics, SPIE, 2020.

[149] Qingfu Zhang and Hui Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, 2007.

[150] Geert Litjens, Robert Toth, Wendy van de Ven, Caroline Hoeks, Sjoerd Kerkstra, Bram van Ginneken, Graham Vincent, Gwenael Guillard, Neil Birbeck, Jindang Zhang, Robin Strand, Filip Malmberg, Yangming Ou, Christos Davatzikos, Matthias Kirschner, Florian Jung, Jing Yuan, Wu Qiu, Qinquan Gao, Philip "Eddie" Edwards, Bianca Maan, Ferdinand van der Heijden, Soumya Ghose, Jhimli Mitra, Jason Dowling, Dean Barratt, Henkjan Huisman, and Anant Madabhushi. Evaluation of prostate segmentation algorithms for mri: The promise12 challenge. *Medical Image Analysis*, 18(2):359–373, 2014.

[151] Yulong Wang, Haoxin Zhang, and Guangwei Zhang. cpso-cnn: An efficient pso-based algorithm for fine-tuning hyper-parameters of convolutional neural networks. *Swarm and Evolutionary Computation*, 49:114 – 123, 2019.

[152] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *NIPS 28*, pages 1135–1143. 2015.

[153] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *4th International Conference on Learning Representations, ICLR, San Juan, Puerto Rico, May 2-4*, 2016.

[154] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML, Lille, France*, volume 37, pages 448–456, 2015.

[155] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[156] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[157] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[158] Bart Van Merriënboer, Olivier Breuleux, Arnaud Bergeron, and Pascal Lamblin. Automatic differentiation in ml: Where we are and where we should be going. In *Advances in neural information processing systems*, pages 8757–8767, 2018.

[159] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML-10, Haifa, Israel*, pages 807–814, 2010.

[160] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *ICML '07*, page 473–480, New York, NY, USA, 2007.

[161] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[162] H Larochelle, D Erhan, and A Courville. icml2007data, Apr 2007.

[163] Seyedali Mirjalili. Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems. *Neural Computing and Applications*, 27(4):1053–1073, 2016.

[164] Seyedali Mirjalili, Seyed Mohammad Mirjalili, and Andrew Lewis. Grey wolf optimizer. *Advances in Engineering Software*, 69:46 – 61, 2014.

[165] James Lin, Kevin Kilgour, Dominik Roblek, and Matthew Sharifi. Training keyword spotters with limited and synthesized speech data. In *ICASSP 2020 - 2020 IEEE In-*

ternational Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 7474–7478, 2020.

[166] Marina Yusoff, Mohamad Syafiq Irfan Saaidi, Amirul Sadikin Md Afendi, and Azrin Mohd Hassan. Tuberculosis x-ray images classification based dynamic update particle swarm optimization with cnn. *Journal of Hunan University Natural Sciences*, 48(9), 2021.