Search Space Optimization of CNN Architecture Using Evolutionary Algorithm

A thesis submitted to the

UPES

For the Award of

Doctor of Philosophy

In

Computer Science

By

VIDYANAND MISHRA

March 2024

Supervisor

Dr. LALIT KANE

School of Computer Science (SOCS)

UPES

Dehradun- 248007: Uttarakhand

Search Space Optimization of CNN Architecture Using Evolutionary Algorithm

A thesis submitted to the

UPES

For the Award of

Doctor of Philosophy

In

Computer Science

By

VIDYANAND MISHRA

SAP ID (500064225)

March 2024

Supervisor

Dr. LALIT KANE

Associate Professor, School of Computer Science

UPES

School of Computer Science (SOCS)

UPES

Dehradun- 248007: Uttarakhand

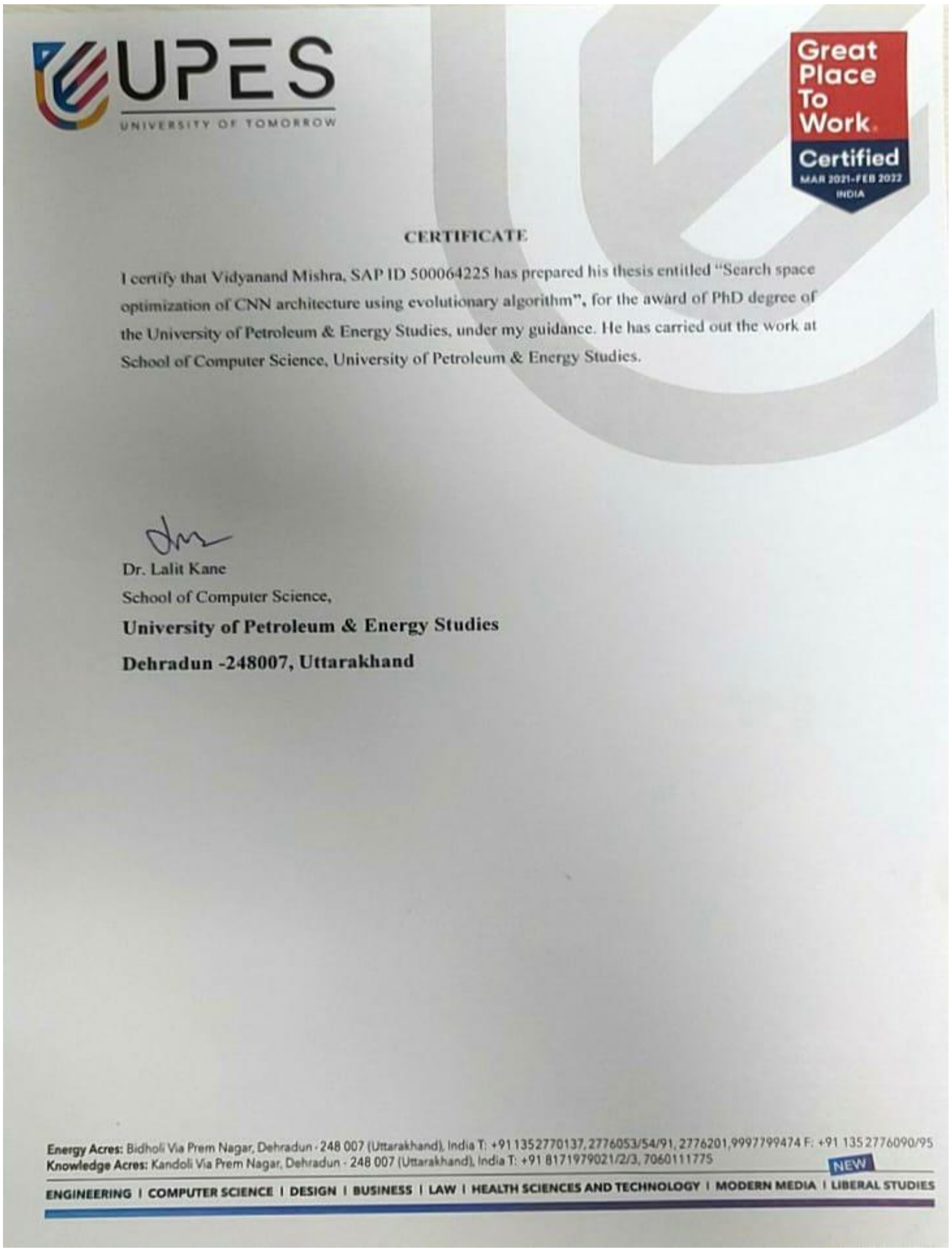
DECLARATION

I declare that the thesis entitled "**Search Space Optimization of CNN Architecture Using Evolutionary Algorithm**" has been prepared by me under the guidance of Dr. Lalit Kane, Associate Professor, School of Computer Science, UPES. No part of this thesis has formed the basis for the award of any degree or fellowship previously.

[illegible]

VIDYANAND MISHRA
School of Computer Science,
UPES,
Bidholi via Prem Nagar, Dehradun, UK, INDIA

CERTIFICATE



UPES

UNIVERSITY OF TOMORROW

Great Place To Work. Certified MAR 2021-FEB 2022 INDIA

CERTIFICATE

I certify that Vidyanand Mishra, SAP ID 500064225 has prepared his thesis entitled "Search space optimization of CNN architecture using evolutionary algorithm", for the award of PhD degree of the University of Petroleum & Energy Studies, under my guidance. He has carried out the work at School of Computer Science, University of Petroleum & Energy Studies.

[illegible]

Dr. Lalit Kane

School of Computer Science,

University of Petroleum & Energy Studies

Dehradun -248007, Uttarakhand

Energy Acres: Bidholi Via Prem Nagar, Dehradun - 248 007 (Uttarakhand), India T: +91 1352770137, 2776053/54/91, 2776201,9997799474 F: +91 135 2776090/95
Knowledge Acres: Kandoli Via Prem Nagar, Dehradun - 248 007 (Uttarakhand), India T: +91 8171979021/2/3, 7060111775 NEW

ENGINEERING | COMPUTER SCIENCE | DESIGN | BUSINESS | LAW | HEALTH SCIENCES AND TECHNOLOGY | MODERN MEDIA | LIBERAL STUDIES

ABSTRACT

The Convolutional Neural Network (CNN) is a complex architecture that performs magnificently in image classification and segmentation problems. Still, selecting an effective architecture is typically hindered by several parameters. Empirically, evolutionary algorithms (EA) have been found adequate in parameter selection and automated neural network search. However, the huge computational requirements imposed by evolutionary search make its applicability unexplored. Consequently, the idea of a CNN architecture selection based on EA is challenging as comparing complex candidate architectures towards their fitness would involve massive computations. This study introduces a novel encoding technique that effectively represents complex convolutional neural network (CNN) architectures. The article provides a definition of fundamental components used to depict the architecture of a Convolutional Neural Network (CNN), including the genesis block, transit block, agile block, and output block. The encoding structure employed in this study facilitates the generation of chromosomal structures with varying lengths.

These structures are initiated through the utilization of evolutionary algorithms. A comparative study is offered to evaluate the effectiveness of the encoding representation in comparison to existing methods. The assessment is predicated upon various aspects, including the quantity of encoding parameters, the expenditure associated with training, and the level of efficiency. We propose a novel framework using an adapted Genetic Algorithm (GA) that automatically evolves an effective CNN architecture. To enhance the effectiveness of the genetic algorithm (GA), we address several key aspects. Firstly, we improve the encoding scheme to better represent the solutions within the genetic framework. Additionally, we refine the process of initializing the initial population, ensuring that it is well-suited to the problem domain. Furthermore, we implement a method for generating diverse offspring, which helps explore the solution space more thoroughly. Moreover, we propose an optimized fitness function that is finely tuned to the problem at hand. This tailored fitness function is designed to accelerate convergence while minimizing the risk of getting trapped in local optima. By strategically adjusting the fitness landscape, we aim to guide the evolutionary process towards more promising regions of the solution space, thereby enhancing the overall performance of the genetic algorithm. The method is validated with the benchmark MNIST, Fashion-MNIST, and CIFAR-10 datasets. The results are comparable to the best manual and automatic state-of-the-art architectures regarding accuracy, convergence rate, and consumed computation resources.

ACKNOWLEDGMENT

Ph.D. journey is truly a life-changing experience and a confluence of multiple learning for me. It is not possible to do without the support and guidance that I received from many people. I am extremely thankful to all of them. I express my heartfelt gratitude to my research supervisor Dr. Lalit kane for his unconditional support and motivation during the entire research program. His guidance always lent a hand to me not only in research but in real life also. His simplicity, positivity, honesty, and discipline inspire me in every second of my life. I wish and try to be as honest and disciplined as he is, for my entire life. Without his continuous support, guidance, and constant feedback, this Ph.D., would not have been achievable.

I express my sincere gratitude to the Honorable Chancellor Dr. Sunil Rai, Honorable Vice Chancellor Dr. Ram Sharma, Registrar Mr. Manish Madan, Dean of the School of Computer Science Dr. Ravi S. Iyer. I am deeply thankful for the resources, opportunities, and academic freedom provided by UPES under your leadership. I greatly acknowledge all the panel members and reviewers who helped me in my research work with their valuable suggestions.

I gratefully acknowledge, Dr. Vishwas Rathi, Dr. Deepa Joshi, Dr. Kiran Kumar Ravulakollu, Dr. Pushpendra Kumar, Ms. Shahina, and Dr. Amit Singh, for their much need support during this Ph.D. program. They were always helpful and provided me with assistance throughout my research journey. I am blessed with a beautiful and supportive family, I am unable to express my gratitude in words to my father Mr. D P Mishra, mother Mrs. Pushpa Mishra, and whole faimly members for always believing in me and encouraging me to follow my dreams.

Above all, I owe it to all almighty for granting me the wisdom, health, and strength to undertake this research task and to shower his blessings on me to complete my thesis.

[illegible]

VIDYANAND MISHRA

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS

Acronym	**Meaning of Abbreviation**
AI	Artificial Intelligence
AE	Auto Encoder
AP	Average Precision
CNN	Convolutional Neural Networks.
COCO	Common Objects in Context
DE	Differential Evolution
DBM	Deep Boltzmann Machines
DCNN	Deep Convolutional Neural Network
DL	Deep Learning
DNN	Deep Neural Network
EA	Evolutionary Algorithm
EP	Evolutionary programming
ES	Evolution strategies
FRRN	Full Resolution Residual Networks
GA	Genetic Algorithm
GP	Genetic Programming
GPU	Graphics processing unit
ILSVRC	Image Large Scale Visual Recognition Challenge
LSTM	Long Short Term Memory
MA	Memetic Algorithm
MLP	The Multilayer Perceptron
NAS	Neural Architecture Search

NN	Neural Network
PSO	Particle Swarm Optimization
RBM	Resitricted Boltzmann machine
ReLU	Rectified Linear Unit
RU	ResNet Unit
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
SD	Standard Deviation

CHAPTER -1

INTRODUCTION

Convolutional Neural Networks (CNNs) belong to the class of supervised deep learning algorithms [8]. The aforementioned algorithms have demonstrated exceptional proficiency in various domains such as computer vision, speech recognition, visual or image recognition, image segmentation, and classification issues [9-11]. The architectural design of the Convolutional Neural Network (CNN) is composed of several key components, namely the convolutional layer, pooling layer, fully connected layer, and their intricate interconnections. Each layer is defined with some fixed set of operations to train the features from input training data, which helps to predict the output from the test data set. The convolutional layer's primary functions are feature extraction, while the pooling layer is used to remove redundant training information. The fully connected layer is utilized to compress the features and predict the output based on the number of classes. The CNN's efficiency depends on factors such as architecture selection [12-13] and hyperparameter selection [14-19]. The range of parameters in the architecture increases with model complexity, such as depth and width. Therefore, it requires more time and resources to train the complex models. The dataset size, pixel size, number of classes, and distribution play a crucial role in designing and training an efficient model. The performance of the model may suffer from overfitting if the training dataset is small and may suffer from underfitting in a linear model. It is particularly challenging to collect large and equally balanced datasets in some domains such as medical and defense; consequently, we also require designing a CNN architecture that works effectively with adaptive data sets.

In CNN, we pass input image as raw pixel data, which helps to learn the model using feature extraction in different layers, as shown in Figure 1.1. The CNN architecture representation comprises a variety of combinations of convolutional layers, pooling layers, and fully connected layers. The convolutional layers consist of multiple two-dimensional matrices known as filters that can update using backpropagation on every iteration. Initial few convolutional layers are accountable for identifying low-level features of images like vertical and horizontal edges. In this layer, filters slide horizontally and vertically to scan the input images, and the weight matrices filters is multiplied with the input image in convolution operation. After that, the value of the filters is modified with each backpropagation operation using a gradient value. After convolution operation, the pooling layer is used, where we apply various mathematical functions to reduce the feature value of the input matrix's selected size and shift based on stride value. It helps to reduce redundancy using various functions max, min, and average and recombine minor features into a major one. After that, the activation function is used to increase the nonlinearity. As per requirement, multiple activation functions such as tanh [20], ReLU [21], and sigmoid [22] are used on the different middle and last convolutional layers. The flattened output from the preceding pooling or convolutional layer is thereafter transmitted to the fully connected layer. The dimensionality of images is reduced through the utilization of fully connected layers, which effectively decrease the amount of pixels involved in convolutional-max pooling procedures. The last layer, subsequent to the fully connected layers, utilizes the softmax activation function [23] to ascertain the likelihood that the input is associated with a particular class (classification). After obtaining the output predictions, a loss function is computed to quantify the discrepancy between the predicted outputs and the actual values. The network is subsequently trained by the utilization of backpropagation, a technique in which the error is propagated in reverse through the layers. The weights of the network are then changed using optimization techniques such as stochastic gradient descent (SGD) or its various adaptations. The process of forward pass (computing the output) and backward pass (updating the weights) is repeated iteratively until the network converges or a predefined stopping criterion is reached. Convolutional Neural Networks (CNNs) are utilized in other domains beyond image classification. These domains encompass speech recognition, natural language processing, video processing [24], image enhancement, and image segmentation [25].

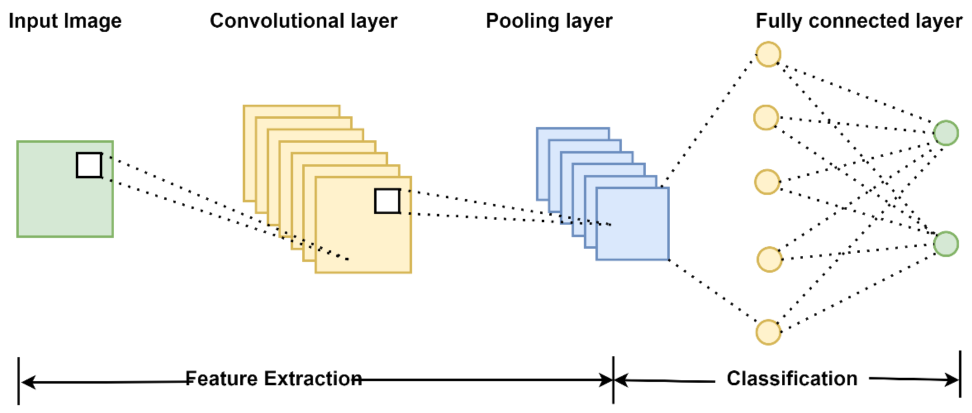


Figure 1.1 Representation of CNN architecture, including convolutional layer, pooling layer, the fully connected layer where a combination of convolution and pooling layer represents feature learning and fully connected is used for classification

1.1 Motivations

CNN architectures consist of several hyperparameters, which need to be tuned and optimized to construct an optimal architecture. The hyperparameters comprise a range of criteria, including the total number of filters, dimensions of the filters, the magnitude of the stride, the kind of pooling function, the choice of activation function, the learning rate, and several more. The task of selecting hyperparameters is a computational challenge, as it falls within the category of NP-hard problems [26]. This challenge arises due to the vast array of potential values and the presence of several parameters involved in the selection process. The achievement of an ideal architecture necessitates substantial human involvement, specialized knowledge in the relevant field, and a process of experimentation and refinement in the selection of hyperparameters. An automatic model is needed to design the architecture based on the input data values and efficiently identify a set of hyperparameters and their values. In order to assess the effectiveness of a Convolutional Neural Network (CNN) architecture using specific hyperparameters, it is customary to train and test the network on a substantial dataset. The computing cost and time required for this procedure might be significant, hence imposing constraints on the number of evaluations that can be executed within a metahurestic algorithm.

1.2 Problem Descriptions

In recent years, there has been a growing body of research [27-32] focused on the automated selection of Convolutional Neural Network (CNN) architectures for image classification tasks. This article primarily focuses on several evolution-based algorithms [7] utilized for the purpose of creating and training from the ground up. In recent studies, few articles on NAS [33] and ENAS [34] have worked on the same domain. However, they primarily focus on comparing architecture selection techniques rather than optimization of training and hyper-parameter algorithms. The researchers have started working on heuristic-based algorithms to rectify these issues. Various heuristic-based evolutionary methods, including genetic algorithm (GA) [35], particle swarm optimization (PSO) [36], genetic programming [37], and differential evolution (DE) [38], have been employed to address the issue of hyper-parameter optimization. These population-based algorithms work on the concepts of the biological behavior of evolution. When using evolutionary algorithms in this domain, we require some encoding technique to represent the CNN architecture in terms of the input population. The classification of encoding techniques is based on two types, namely fixed-length and variable-length encoding schemes. The fixed-length encoding strategy is a straightforward method to execute when the maximum depth of a convolutional neural network (CNN) is predetermined. However, it has the potential to limit the extent of exploration [39-41]. The depth is adaptive in the variable-length encoding scheme, which means it can explore wider depending on requirements. Therefore variable length encoding scheme is more suitable in architecture evolution. Additionally, we must define all the functions again as per the encoding. The population is initialized randomly in an encoding scheme that makes it more generic in available search space and easy to use. After that, the encoded architecture is passed to an algorithm evaluated based on the dataset and fitness function. In this research accuracy (1-RMSE) is considered as a fitness function. Based on the fitness value, a few CNN architectures are selected to have high accuracy for the next iteration. The next iteration generates the new population using mutation and crossover operators defined in the algorithm. It helps to identify new populations without stuck in local optima. After multiple iterations, this process will converge the automatically generate the most suitable CNN architecture in defined constraints.

1.2.1 Hyperparameter Tuning

CNN architectures typically have numerous hyperparameters, such as the number of layers, filter sizes, learning rates, sample sizes, etc. The high dimensionality of the hyperparameter space makes the search process more challenging and computationally expensive.

1.2.2 Expensive Evaluation

In order to assess the effectiveness of a Convolutional Neural Network (CNN) architecture using specific hyperparameters, it is customary to train and test the network on a substantial dataset. This process can be computationally expensive and time-consuming, limiting the quantity of assessments that can be undertaken within a genetic algorithm.

1.2.3 Overfitting and Generalization

Convolutional Neural Network (CNN) architectures have a tendency to exhibit overfitting, a phenomenon characterized by the model's ability to achieve high performance on the training dataset but perform badly on unseen or test data. Relying exclusively on the performance of the training data for the selection of hyperparameters may result in unsatisfactory generalization. It is imperative to exercise caution in order to ascertain that the chosen hyperparameters exhibit strong generalization capabilities when applied to data that has not been previously seen.

1.2.4 Search Space Exploration

CNN architectures can have a vast search space with a multitude of possible hyperparameter combinations. Genetic algorithms need to effectively explore this space to find optimal or near-optimal solutions. However, due to the high dimensionality and complex interactions, it can be challenging to achieve thorough exploration within a reasonable computational budget.

1.3 Objectives

Propose a methodology to design an effective CNN model using evolutionary algorithms for the given dataset.

Sub-Objectives

1. Design an encoding strategy to represent CNN architecture building blocks and their interconnections.
2. Propose a method to fine-tune hyperparameters of the CNN model using evolutionary algorithms for the given dataset.
3. Comparative analysis of the proposed CNN model with the existing state-of-the-art techniques in terms of classification accuracy and number of parameters.

1.4 Contribution of the Thesis

The general goal of this thesis is to propose a methodology to design an automatic CNN model using genetic algorithms using a given dataset. The research contribution is summarized below.

To begin we proposed scheme employs a variable-length encoding scheme that represents the depth as well as the width of the architecture. The main advantage of the proposed encoding scheme is that it can represent architecture with a combination of two different layers. It makes the representation simple and one can increase the depth of architecture easily. Also, due to fewer parameters, one can define different evolutionary operations like mutation and crossover efficiently. The scheme also supports increasing the complexity within the block. In the agile block, it can generate filter size and depth randomly and thereby increases complexity. The proposed scheme supports a hybrid-encoding scheme that utilizes binary as well as decimal representation. The encoding scheme offers the maximum choice of exploration in depth and width as well as faster optimization. We pass our initialized encoding method in evolutionary algorithms to optimize for better architecture.

Second, we proposed a novel framework to evolve CNN architectures using genetic algorithms. The process encompasses the exploration and refinement of the architecture's hyperparameters, which encompass various aspects such as the number of layers, types of layers, sizes of filters, and patterns of connectivity. By iteratively applying selection, genetic operators, and evaluation, the genetic algorithm explores the search space of CNN architectures, favoring the ones that perform better on the defined metrics.

Third, the proposed methodology applies the population initialization, fitness calculation and offspring generation of the CNN architecture. Initially, the number of population and the depth of each population is selected randomly. In the selected population, the first layer is fixed as a convolutional layer; then, convolutional and pooling layers are determined randomly with equal probability. The convolutional layer's filter count is randomly chosen in the given range. An individual's CNN is initially decoded using a predetermined set of hyperparameter parameters. CNN decoding is trained with training data, and accuracy is used to determine fitness. Because the training of CNN is a time-taking task, we used half of the dataset for initial training to make it efficient. For offspring, generation two parents are selected based on which of two randomly selected individuals is more suitable. We build a new set of populations with equal probability by utilising mutation and crossover processes. The purpose of mutation is to explore the search space by introducing small random perturbations to the existing architectures. Mutation helps to prevent premature convergence and allows the algorithm to potentially discover novel and beneficial architectures. The probability of mutation determines how frequently mutation is applied. A higher mutation rate increases the chances of mutation occurring. The objective of crossover is to generate offspring that acquire advantageous characteristics from both progenitors, hence potentially resulting in enhanced designs. The location(s) or process at which genetic material is exchanged between parental organisms is referred to as the crossover point(s). Various crossover tactics can be utilized, including single-point crossover, uniform crossover, and multi-point crossover.

The proposed methodology is assessed and contrasted with 11 contemporary peer contenders, comprising of four partial tuning and seven automatic algorithms for determining the architectures of CNNs. The empirical findings obtained from conducting experiments on the MNIST, Fashion_MNIST, and CIFAR10 datasets demonstrate that the proposed methodology has the capability to autonomously create deep convolutional neural network (DCNN) architectures that are on par with, or perhaps beyond, the most advanced models currently available.

1.5 Thesis Outline

The following is an outline of the thesis. The second chapter delves into the fundamental components of computer vision technologies, starting with manual deep convolutional neural network designs employed in image classification. The different evolutionary algorithm based CNN architecture and MNIST, Fashion_MNIST, and CIFAR10 datasets are introduced in Chapter 3. A novel framework for designing CNN architecture using genetic algorithm is put forward in Chapter 4. In Chapter 5, we discussed the hyperparameters required to fine-tune the model. Furthermore, the suggested architecture's performance is evaluated and compared to existing approaches based on accuracy, epoch, number of generation and GPU days required. In Chapter 6, we summarize the thesis, reach conclusions, and talk about future research.

CHAPTER -2

LITERATURE SURVEY

The convolutional neural network is structured with a layered architecture that includes a convolutional layer, a pooling layer, and a fully connected layer. The raw pixel data from the input image is transmitted to a Convolutional Neural Network (CNN), facilitating the extraction of features at several levels to enhance the learning process of the model. Weighted filters are employed within the convolutional layer to extract distinctive characteristics from the input data, while the activation function is utilized to add nonlinearity. The pooling layer serves the purpose of eliminating redundant features from the convolutional layer by the utilization of procedures such as minimum, maximum, or average. The resulting matrix is converted into a unidimensional vector and utilized as input for a fully connected neural network layer during the training process.

Numerous hyperparameters must be altered and enhanced to improve the CNN model. The filter size, the number of filters, pooling function, learning rate, activation function, stride size, and many more hyperparameters were among them. Due to huge parameters, researchers working on generating a CNN architectural design need help choosing appropriate hyperparameters. The complexity of the CNN architecture is a critical factor while learning complicated features from training datasets. As an architecture's depth and interconnections increase, so does its parameters and complexity. We need methods to automatically discover the hyperparameters and CNN architecture to solve the problem.

The performance of CNN architecture is determined by accuracy, training cost, and parameter count. The accuracy is mostly determined by the training dataset (image size, quality, and distribution) and the complexity of the architecture. However, the training cost is mostly associated with parameters such as depth of architecture, size of kernels, number of kernels, learning rate, epoch, activation function and many more. Hence, selecting an accurate architecture is tedious, as it takes knowledge of the CNN domain and several trial-and-error combinations for hyperparameter tuning.

In recent years, several fascinating studies on deep CNNs have been carried out that elaborate on the crucial elements of CNN and their alternatives. There exist quite a few survey articles on CNN using evolutionary algorithms (EA) that focus on a certain category of optimization problems. The survey conducted in reference [42] highlights the inherent taxonomy observed in manually designed deep convolutional neural network (CNN) architectures. Consequently, the CNN architectures are categorized into seven distinct groups. The fundamental principles behind these seven categories encompass spatial exploitation, depth perception, multi-path propagation, breadth of coverage, feature-map utilization, channel enhancement, and attentiveness. This study has identified a significant difficulty, namely that even minor adjustments to hyper-parameter values can have a substantial influence on the overall performance of a Convolutional Neural Network (CNN). Consequently, the meticulous choice of hyper-parameters emerges as a significant aspect of the design process, necessitating the implementation of a suitable optimization approach. In recent works, EA is gaining momentum for architecture selection and hyperparameter optimization in generic deep neural networks (DNN) models. A survey [43] described the EAs approach to designing and configuring deep neural network architecture. It has highlighted the strength of EAs in investigating the challenges involved in suitable DNN architecture design and training. The authors included designing and training different DNNs architectures, including CNNs, Recurrent neural networks (RNNs), Deep Boltzmann Machines (DBMs), Resitricted Boltzmann machines (RBMs), and Auto encoders (AEs). The paper includes the effectiveness of EAs in deep learning; however, more explanation is still needed why some EAs algorithms perform better than others.

2.1 Evolution of CNN Architetcure

The inception and advancement of Convolutional Neural Network (CNN) architecture began in the 1980s. The initial proposal for a multilayered architecture, known as ConvNet, was put up in 1989 by LeCun. Numerous architectures were proposed after ConvNet by modifying the existing architecture. However, it could not solve real-life problems due to limited computation power and the limited dataset. In 1998, LeCun proposed a five-layered architecture known as LeNet-5 [44], primarily used for digit classification. However, it could not gain much popularity in the different domains because the bounded computational power and the standard datasets are unavailable. After a decade, AlexNet [45] (a variant of CNN) brought the main breakthrough in the performance of the CNN architecture in ILSVRC (Image Large Scale Visual Recognition4 Challenge) in 2012. From 2012 until now, several attempts have been conducted to improve the accomplishment of CNN architecture. Several state-of-the-art convolutional neural network (CNN) architectures have been proposed, including VGG Net [46], ResNet [47], GoogleNet [48], XceptionNet [49], and others. These architectures have demonstrated significant advancements in image classification tasks through various approaches, such as increasing the depth or width of network layers or modifying the network parameters.

The LeNet architecture emerged as the early convolutional neural network (CNN) model that shown promising outcomes in effectively addressing the challenge of digit recognition. The previous architectures considered each image pixel as a single unit for input in CNN architecture, which increased the number of input parameters and made it less efficient. LeNet has worked on this limitation by evaluating the correlation between neighboring pixels, reducing the number of parameters in the convolution operation. LeNet architecture was simple and linear, restricting its effectiveness to simple grayscale digit classification problems instead of generic image classification tasks. Krichevsky et al. proposed AlexNet architecture in 2012. It enhances the feature extraction capability of CNN architecture by making it deeper into up to 8 layers. However, as the depth increases, the number of parameters increases, making architecture more complex. Several problems arise with the complexity of architecture and smaller datasets, such as the overfitting and vanishing gradient. AlexNet solved the problem of overfitting up to some level by increasing the depth of the architecture using the concept of random skip connections. Additionally, it uses a ReLU activation function to reduce the vanishing gradient problem [50]. It uses a large filter size (11×11, 7×7, and 5×5) to improve the learning accuracy.

Simon Yan et al. proposed a 19-layer deep architecture named VGGNet. The problem of AlexNet is complex heterogeneous filter size (7×7, 11×11, 5×5) in different layers of the CNN architecture. VGGNet architecture has solved this problem by keeping filter size 3×3 to make it homogenous. It improves the computation cost with similar accuracy by reducing the calculations involved. VGGNet became popular because of its homogenous structure. It also uses a new max pool operation in the pooling layer. VGGNet architecture resulted in improved performance concerning image classification tasks by incorporating a large number of parameters. It is still not suitable for low resource systems due to the involvement of increased computational cost. The number of parameters in VGGNet is approximately 138+ million, making it computationally high. To solve this problem, GoogleNet introduced a new inception block that replaced the existing convolution layer. It uses 1×1 convolution to optimize the number of matrix multiplications. It also removes the fully connected layer and uses the concept of sparse connections with average pooling to reduce the number of parameters. It reduces the total parameters from 138 million to 4 million with promising efficiency. The main limitation of Google Net is the heterogeneous structure of the filters in the inception block.

There was an early perception that the CNN architecture's depth closely correlated with the architecture's ability to learn. With the depth of architecture, vanishing gradient problems arise. Therefore, training deep neural networks is challenging due to the vanishing gradient problem. The ResNet architecture was developed in 2016 to reduce the vanishing gradient issue with increased depth to solve that problem. In ResNet architecture, a residual block is introduced, employing the concept of skip connection. ResNet introduces a 50/101/152 depth architecture that performs remarkably in the benchmark image dataset COCO [50]. However, due to the huge depth and complex architecture, the number of parameters increases linearly, not exponentially. As part of architecture design and to address the issue of vanishing gradients in deep CNN architectures, DenseNet [51] was proposed in 2017. The gradient propagation and classification accuracy are improved by creating shorter paths between a layer and its subsequent layer closer to the output. It is used to connect every layer with the next subsequent layers directly, which helps to concatenate features directly. Supposing L is the total number of layers in the architecture, then a total of L (L+1)/2 connections will be possible in the DenseNet. Table 2.1 illustrates the comparative analysis of the performance of various CNN architectures designed manually. The Accuracy of the CNN architecture depends on numerous factors such as topology (convolution layers, pooling layers, and their interconnections) and learning parameters (including weight, bias, and hyperparameters like activation function, filters size, number of filters, and many more).

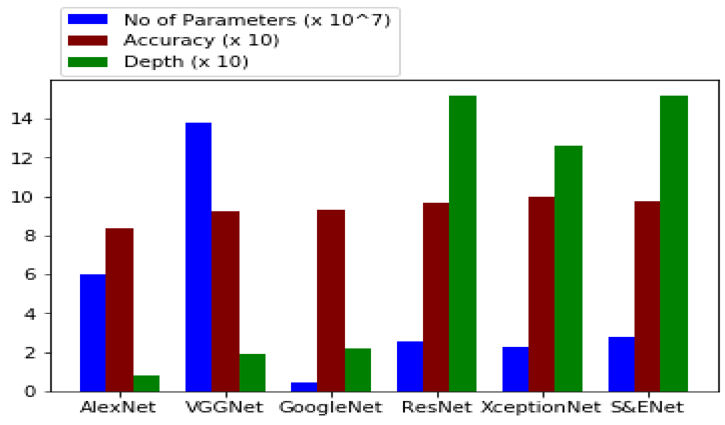


Figure 2.1 Comparisons of various CNN architectures under the ImageNet dataset

Table 2.1 Manually designed CNN architectures are compared based on parameters, accuracy and depth.

Reference and Year	**Architecture Name**	**No. of Parameters (in millions)**	**Data Set**	**Accuracy/ Error**	**Depth**	**Relevant Review Findings**
[44], 1998	LeNet	0.060	MNIST	0.95	5	▪ First effectively designed CNN architecture in the real-time data set. ▪ Large filter sizes are used.
[45], 2012	AlexNet	60	ImageNet	16.4	8	▪ It increases the depth of architecture for real-time RGB ImageNet data set. ▪ ReLU activation function is implemented for enhancing non-linearity. ▪ Dropout function is applied to reduce the model complexity.
[46], 2014	VGG	138	ImageNet	7.3	19	▪ The architecture is made complex by increasing the number of parameters to enhance the accuracy. ▪ Homogenous topology is observed throughout the architecture. ▪ Filter Size is reduced to minimize the computation of the model.
[48], 2015	GoogLe Net	4	ImageNet	6.7	22	▪ It introduces the concept of blocks in the CNN architecture. ▪ Apply network in network concept. ▪ Used 1×1 convolutional filter
[47], 2016	ResNet	25.6 1.7	ImageNet CIFAR-10	3.6 6.43	152 110	▪ Introduced skip connection to minimize the size of the total parameter. ▪ Residual learning is used.

[98], 2016	Wide ResNet	36.5	CIFAR-10 CIFAR-100	3.89 18.85	28 -	▪ The width of the architecture has been increased, resulting in a more complex architecture.
[49], 2017	Xception	22.8	ImageNet	0.945	126	▪ Convolution options are divided into two categories in this architecture. The depth-wise convolution method is used first, followed by the point-wise convolution method.
[101], 2017	Squeeze & Excitation Networks	27.5	ImageNet	2.3	152	▪ Used SE block before convolution layer to suppress less important feature. ▪ Adapts the feature maps of each layer.
[51], 2017	DenseNet	25.6 25.6	CIFAR-10 CIFAR-100	3.46 17.18	190 190	▪ Multiple layers are connected together for Cross-layer information flow. ▪ The amount of feature maps at each layer has resulted in a significant rise in parameters. ▪ Decision layers can access characteristics at both the low level and high level.
		15.3 15.3	CIFAR-10 CIFAR-100	5.19 19.64	250 250	

However, in Figure 2.1, the main concern is discussed that the association between accuracy with the number of parameters and the depth of CNN architecture is abrupt. Initially, from AlexNet to VGGNet architectures, the architecture complexity increased with depth, increasing the number of parameters. Therefore the vanishing gradient problem occurs. Due to the vanishing gradient, the model's accuracy is declining even with deeper and more complex architecture. ResNet and GoogleNet applied the skip connection concept that reduces the number of parameters and minimizes the vanishing gradient problem. Another concern is that the selection of parameters and their value is also challenging as the number of parameters is huge and different in nature. However, selecting parameters for the suitable architecture design is still challenging. Manually identifying the appropriate number of parameters is extremely difficult as the number of parameters is huge, and its relationship with accuracy is uncertain. The metaheuristic optimization algorithms assist us in finding a suitable solution for the above problem as those algorithms are gradient-free and can escape to local optima.

2.2 Hyperparameter Selection

Despite the fact that CNN is quite powerful, its effectiveness or accuracy is dependent on the parameters selection. When selecting the convolutional neural network parameters, the optimal combination of several parameters is usually used. The CNN model's efficiency is determined by two major factors. The first is architecture selection, which includes architectural depth, the number of layers (such as a convolutional layer, pooling layer, fully connected layer, and their interconnection). Selection and initialization of hyper-parameters such as the number of kernels, kernel size, stride, padding, pooling operation, optimizer, activation function, number of hidden layers, dropout, batch normalisation, learning rate, loss function, embedding vector size, and epoch number are also essential. The kernel, a 2-D weight matrix, is used for feature extraction in the convolutional layer. The convolution process compresses the input matrix. To address these issues, we can add rows and columns to the input image before convolution, which is known as padding. The amount of pixels we will jump when we convolve the filter/kernel is determined by stride, which we used when doing the convolution procedure. In addition to convolutional layers, pooling layers are frequently used by CNNs to reduce input size, accelerate computation, and strengthen some of the features it detects. We can use max, average, or min pooling operation in the pooling layer. After selecting CNN architecture (layers and their connection), a few other hyperparameters are required to set before training. The following are as follows:

2.2.1 Learning rate

The optimization algorithm's learning rate determines how frequently the weight is updated. We can select a constant learning rate, a learning rate that gradually declines, momentum-based approaches, or adaptive learning rates depending on the optimizer we use.

2.2.2 Number of epochs

The number of epochs refers to how frequently the neural network processes the entire training data. We should increase the number of epochs as soon as there is a slight difference between the training and test errors.

2.2.3 Batch size

The batch size specifies how many samples will be sent over the network at the same time. Large data sets necessitate slow CNN training. As a result, it is necessary to seek a faster optimization technique. It is recommended that batch size should be a power of two to run code faster.

2.2.4 Activation function

The activation function of the model introduces non-linearity. Multiple activation functions, such as ReLU [21] in Eq 1, sigmoid [22] in Eq (2), tanh in Eq(3) [20], and others, are used in different layers of CNN architecture. The output of Relu lies in all positive value, sigmoid value lies in range (0,1) whereas tanh lies in range -1 to 1. Sigmoid function mostly uses in outer layer in binary classification problem where as relu and tanh uses in intermediate layers.

$$f(x) = \max(0, x) \quad \dots\dots\dots\dots\dots\dots\dots\dots\dots (1)$$

$$f(x) = 1/(1 + e^{-x}) \dots\dots\dots\dots\dots\dots\dots\dots (2)$$

$$f(x) = ((e^{x} - e^{-x}) / (e^{x} + e^{-x})) \dots\dots\dots\dots (3)$$

2.2.5 Number of hidden layers

It is typically a good idea to keep adding layers until the test error stops improving. The trade-off is the cost of computationally training the network. A small number of units can result in underfitting, whereas a larger number of units is usually not hazardous with correct regularisation.

2.2.6 Dropout

The utilization of dropout [52] has emerged as a popular regularization technique in deep neural networks, serving the purpose of mitigating overfitting. This technique selectively eliminates neural network units based on their probability of removal.

2.2.7 Optimizer

Optimizers are algorithms or techniques that modify neural networks' weights and learning rates to minimize losses. Multiple optimization methods such as SGD [53], Adam [54], Adagrad [55], AdaDelta [56], or RMSProp [57] are used in CNN architecture. Adam is the most effective optimizer if one wants to train the neural network in less time and more efficiently.

2.2.8 Loss function

The disparity between the output generated by the Convolutional Neural Network (CNN) model and the desired output is quantified through the utilization of a loss function within a neural network. The gradients necessary for updating the weights can be derived from the loss function. The cost is calculated by computing the mean of all incurred losses. Within the context of Convolutional Neural Network (CNN) architecture, it is common to employ several well-known loss functions, namely Categorical Cross-Entropy [58], Sparse categorical cross-entropy [59], and Mean squared error [60].

2.3 Data Preprocessing

The complexity of the architecture and the input data impact how well CNN performs. Image preprocessing is needed before being used for model training to improve CNN learning. Images originate from several sources of different sizes and shapes. Making all images the same size is the initial step in data preprocessing. There are multiple methods of image preprocessing available such as transformation, normalization, and augmentations. Any image alteration in form and shape is referred to as morphological transformation. Erosion, dilation, opening, and closing are common morphological transformation methods. Normalization is another preprocessing method for rescaling the pixel value in a confined range. The purpose of doing this is to assist with the gradient propagation problem. Often, the amount of data we have is insufficient to accomplish training deep neural network models. In these situations, we undertake data augmentation to expand the dataset. Only the training set should be used for augmentation, and the validation set should never be used. Augmentation techniques [66] like flipping, rotating, cropping, translating, illuminating, scaling, and adding noise generate additional data from raw data, and significantly enhance the model's accuracy.

CNN architecture solves image classification and segmentation problems in every domain. The model's accuracy depends on architecture selection and the dataset. If a fewer number of images, it may get stuck in underfitting. If the resolution is less, the feature may vanish with the depth of architecture. Therefore, we required high resolution, huge data size, with balance in every class for better performance. Some benchmark datasets developed to compare the performance are shown in Table 2.2 MNIST dataset was the benchmark dataset for digit classification. The MNIST dataset's complexity is less, so it can be easily computed with less computation power. It has ten classes with grayscale images having 60000 training data with pixel size 28×28.

Most models are used as a reference for MNIST until the 2012 ImageNet competition. After that, challenging datasets like CIFAR 10 for classification into ten classes and CIFAR 100 for classification into hundred classes are used.

2.3.1 Dataset

The CIFAR10 dataset depicted in Figure 2.2, in particular, serves as a benchmark for image classification, classifying ten different types of natural objects, including birds, horses, ships, deer, frogs, dogs, trucks, cats, vehicles, and aeroplanes. It comprises 60,000 RGB images, each 32×32 pixels in size. Furthermore, there are 50,000 images in the training set and 10,000 in the testing set. There are the same amount of images in every category. Similarly, the MNIST dataset, shown in Figure 2.3 is a benchmark for image classification for identifying digits. It has 70,000 grayscale pictures in total, each measuring 28×28 pixels. Furthermore, the training set consists of 60,000 images, and the testing set contains 10,000 images. The amount of data in each category is the same. Also, we used Fashion_MNIST dataset Figure 2.4, as a benchmark for fashion image classification. It has 70,000 grayscale pictures in total, each measuring 28×28 pixels. It was created in 2017 and had ten classes. Furthermore, the training set consists of 60,000 images, and the testing set contains 10,000 images. The amount of data in each category is the same.

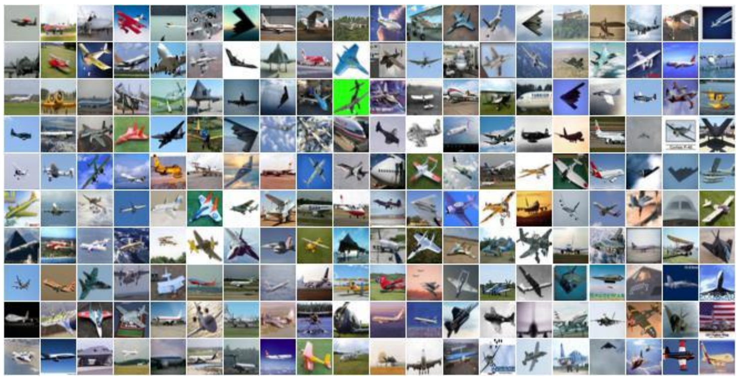


Figure 2.2 Examples from CIFAR-10 data sets

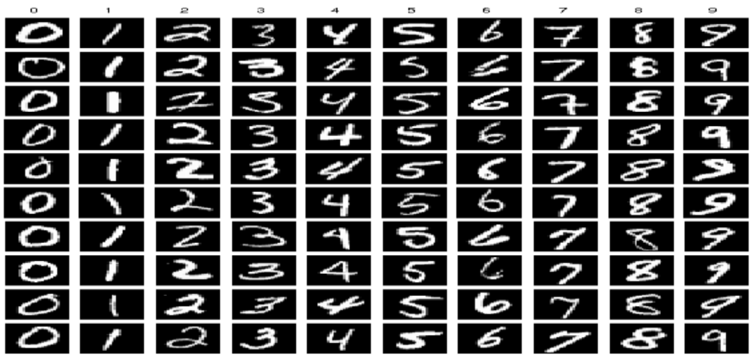


Figure 2.3 Examples from MNIST data sets

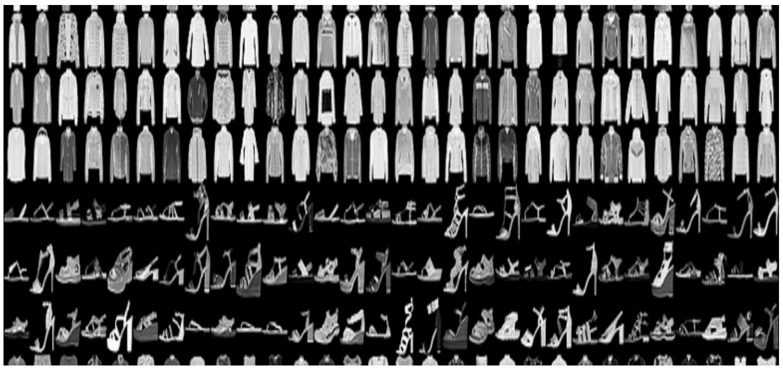


Figure 2.4 Examples from Fashion_MNIST data sets

Table 2.2 Dataset used in different manual and automatic CNN architecture for Image classification

Data set	**Input size (Pixels)**	**No. of classes**	**No. of training images**	**No. of testing images**	**Image type**
MNIST [3]	28 × 28	10	60,000	10,000	Grayscale
CIFAR-10 [2]	32 × 32	10	50,000	10,000	RGB
CIFAR-100 [2]	32 × 32	100	50,000	10,000	RGB
ILSVRC 2012 [61]	224 × 224	1000	1.3M	1,50,000	RGB
Fashion MNIST [62]	28 × 28	10	60,000	10,000	Grayscale

2.4 Optimization algorithms

An optimization can be a minimization or maximization issue of any given function in the defined search space. Numerous categories exist for classifying optimization algorithms, including gradient-based and gradient-free algorithms, deterministic and stochastic algorithms, trajectory-based algorithms, and population-based algorithms. Gradient-based algorithms make extensive use of derivative information to solve differentiable and convex optimization problems. When functions have discontinuities or multiple local optima exist, the gradient-based method is ineffective. For optimizing discontinuous functions, metaheuristic algorithms are more feasible. However, metaheuristic algorithms have some limitations. It is appropriate to find good answers to complicated optimization problems within given constraints, but finding optimal solutions is not guaranteed. There exists no known mathematical model that can solve all types of optimization problems with improved accuracy. For example, structure optimization [63], traveling salesman problem [64], Hamiltonian cycle problem [65], and the halting problem of the Turing machine [66] are some known as NP-complete problem, which is not solvable in polynomial time using a deterministic algorithm. Therefore, approximation algorithms based upon intelligent search are proposed to suggest the correct solution to a particular NP-complete problem.

The parameters chosen and optimized have an impact on how well the CNN architecture operates. Because CNN includes many parameters with different properties, we require optimization methods that can handle both discrete and continuous variables. The metaheuristic algorithms are classified into two categories population-based and nature-inspired as shown in Figure 2.5. Any metaheuristic algorithm will typically consist of two main parts: intensification and diversification. While diversification by randomness enables the search to stray from local optima, the best solutions are chosen to guarantee that solutions will converge to the optimum.

Approximation algorithms start with an initial guess and generate an iterative sequence with improved results. It uses an objective function to mathematically optimize the given problem with bounded variables and their condition until they terminate. The termination condition is based upon the generation number or target value. The optimization strategy moves from one generation to another based on the suggested optimization algorithm. Most of the strategies use the objective function's value to calculate the solution's fitness value, subject to the constraint functions. Approximation algorithms run based on information gathered from the current generation as well as the previous generation. The approximation algorithm begins with a random value and generates the sequence of solutions $x_0, x_1 \dots\dots, x_k$ such that $f(x_{i+1}) < f(x_i)$.

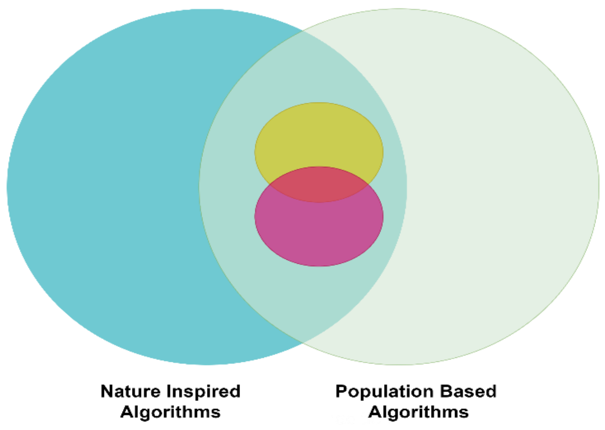


Figure 2.5 Classification of metaheuristic approach of optimization

Many meta-heuristic algorithms [67-68] have been proposed to solve hard mathematical problems in recent decades. The growing power of meta-heuristic algorithms attracted the attention of researchers. Evolution Algorithms [69-73] and Swarm Intelligence [74-76] are four broad categories of meta-heuristic algorithms shown in Figure 2.6. Evolution algorithms are inspired by biological evolution; their formulation is based upon selection, recombination, and mutation. At the same time, swarm intelligence is based upon swarms' collective and independent behavior. Swarm intelligence is a concept that draws inspiration from the intelligent behavior exhibited by swarms. In this paradigm, swarms are capable of collectively identifying solutions by using both local and global best positions, and their velocity is based on distance from the actual object.

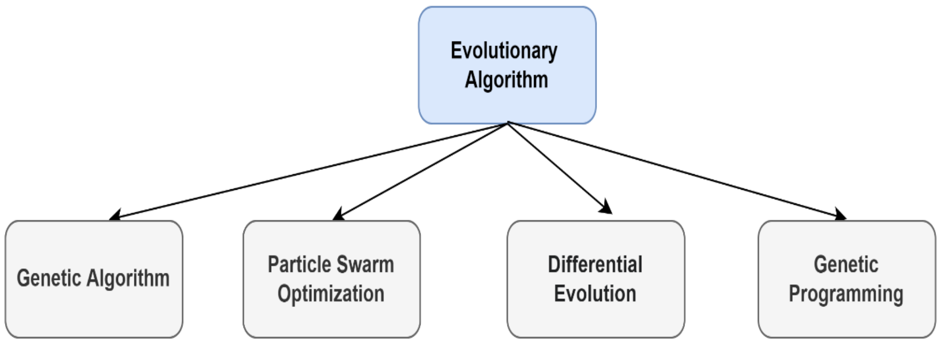


Figure 2.6 Categories of evolutionary algorithms regarding the search strategies

Evolutionary algorithms are used to solve gradient-free optimization problems. It can also be used in a single objective and a multi-objective problem. We discussed some algorithms with their domain used in optimizing CNN architecture based on literature as follows:

2.4.1 Genetic Algorithm

A genetic algorithm (GA) [77] is a subset of evolutionary algorithms helpful for solving multidimensional nonlinear problems. GA is inspired by biological evolution, and its formulation is based upon selection, recombination, and mutation operations. A selection operation is defined as selecting a set of an individual based on a fitness value where a fit individual has more probability of being selected. Recombination or crossover is an operation to generate a new individual by recombining two or more individuals based on some recombination rule. The mutation is a process of introducing diversity in the solution set by generating some set of values around the selected point. GA works by applying “random” changes to current solutions to generate new ones. A fitness function is used to select the best parameters, and solutions representing the higher fitness value are chosen. GA can initialize its population randomly with different positions and start optimizing in multiple directions, making it suitable for multiple-minima problems. It can also help solve the brute force approach and NP-hard problems. The major drawback of this algorithm is that it does not guarantee the optimal solution but is close to the optimal one.

Additionally, its convergence rate becomes slower when it is closer to the optimal solution. Nowadays, GA has become extremely popular because of the neural networks, as the convergence rate of neural networks is slow due to gradient-decent problems. Moreover, it is likely to stick in local minima due to multiple minima where GA can help solve the problem. Figure 2.7 represents the flow chart of the Genetic Algorithm. In the flow chart, different operations like initialization, mutation, and crossover are represented as square boxes, where stopping criteria are defined in a rectangular box.

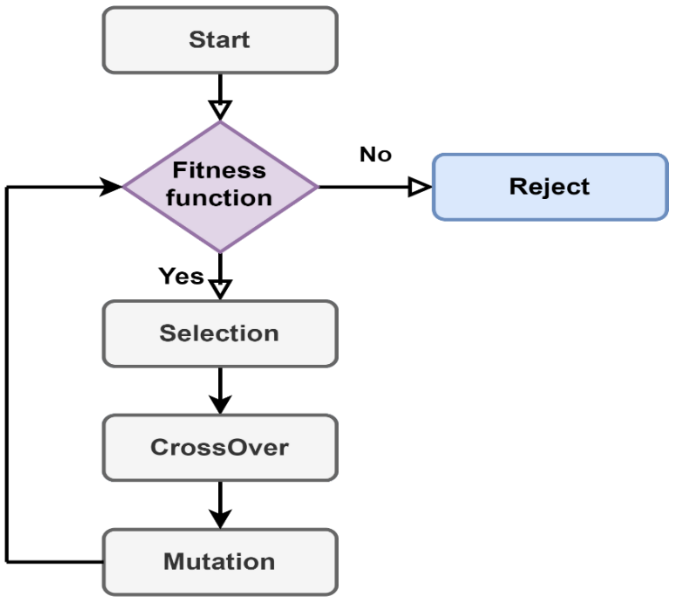


Figure 2.7 Flow chart of genetic algorithm

2.4.2 Particle Swarm Optimization (PSO)

In the Particle Swarm Optimization (PSO) algorithm, each particle is initially assigned a position and velocity. These values can be modified over the iterations of the algorithm, depending on the number of iterations. The rate of change is contingent upon the local acceleration constant c_p and global acceleration constant c_g shown in Eq.1. Another parameter V_i^{t+1} represents velocity in the next iteration, w is the weight value, r_p represents the random value for local convergence in the range [0, 1], similarly, r_g represents global randomly generated value in the range [0, 1] shown in Eq.1. As shown in Eq.2, variable p_i^{t+1} represents the position vector that is updated with the previous position and current velocity. The $p_{best_{id}}$ represents the local best position whereas $g_{best_{id}}$ is the global best position vector. Additionally, as shown in Eq.3, $w(iter)$ is an adaptive weight value that is bounded in the given range $[w_{min}, w_{max}]$. The w_{max} is the maximum weight value, w_{min} is the minimum, $iter_{max}$ is the maximum iteration defined and $iter$ is its current iteration. Our target is to reach the global best from the local best in a minimum number of iterations.

$$V_i^{t+1} = w \times V_i^t + c_p \times r_p \times \left(p_{best_{id}} - p_i^t\right) + c_g \times r_g \times \left(g_{best_{id}} p_i^t\right) \tag{1}$$

$$p_i^{t+1} = p_i^t + V_i^{t+1} \tag{2}$$

$$w(iter) = \left(\frac{iter_{max} - iter}{iter_{max}}\right) \times (w_{max} - w_{min}) + w_{min} \tag{3}$$

2.4.3 Differential Evolution (DE)

Differential evolution (DE) [78] is a heuristic technique that allows nonlinear and non-differentiable continuous space functions to be globally optimized. The advantage of DE algorithms is simplicity and few control variables that need minimal controls to direct the minimization. It has good convergence qualities or the global minimum's constant convergence in successive independent trials. The classic differential evolution (DE) algorithm requires only three control parameters: the scale factor, crossover rate, and population size. Here, an initial population of real-valued decision vectors commonly referred to as chromosomes are randomly chosen to start the process. In DE, the mutation operator is used to produce new suitable solutions during each iteration. Additionally, the crossover technique is used to expand the diversity of the modified parameter vectors.

2.4.4 Genetic Programming (GP)

Genetic programming (GP) [71] uses an algorithm that uses crossover, random mutation, a fitness function, and several generations of evolution to solve the optimization problem. This algorithm is inspired by biological evolution and its basic mechanics. A genetic program is represented as a tree structure of actions and values, which is commonly referred to as a hierarchical data structure. In the field of Genetic Programming (GP), the procedures entail the selection of the most optimal programs for crossover reproduction and mutation, guided by a pre-established fitness measure that is often proficient in the desired task. In the field of genetic programming (GP), the crossover operation is employed to generate new and distinct children by exchanging random segments between selected pairs of parents. These offspring thereafter contribute to the formation of the next generation of programs. Mutation is the term used to describe the process of substituting a randomly selected component within a program with another randomly selected component. In the field of Genetic Programming (GP), the subsequent generation emulates certain programs from the present generation that were not selected for reproduction. Subsequently, the succeeding iteration of programs undergoes cyclic selection and other related procedures.

2.4.5 Evolution Strategies (ES)

Evolution strategies (ES) [70] is a subclass of Evolutionary Algorithms that are nature-inspired direct search and optimization methods. It employs mutation, recombination, and selection on a group of individuals with candidate solutions to develop better solutions iteratively. In contract to optimization, ES is great for non-linear or non-convex continuous optimization issues. Compared to other evolutionary algorithms, population sizes in ES are extremely small. The effectiveness of an evolutionary strategy (ES) in addressing a certain problem type is significantly influenced by its design of the ES-operators used, such as mutation, recombination, and selection, as well as how the ES-operators are altered during the evolution process.

2.4.6 Evolutionary Programming

Evolutionary programming [79] is one of the evolutionary algorithms that allows the numerical parameters of the program to evolve while keeping the program's structure constant. In contrast to conventional genetic algorithms, evolutionary programming focuses on individual behaviors; therefore, the crossover is not used in favor of mutation. The only way to produce individuals is to mutate their parents, and the behavior is defined by the degree of the mutation. Instead of using raw fitness value, relative fitness value is utilized to quantify how much better a person performs than those around them. The most frequent applications of evolutionary programming are in constrained settings like scheduling and routing, power systems, and system design.

2.5 Evolving CNN Architecture using Evolutionary Algorithms

The evolution of CNN architecture begins with layered architecture increases as per the improvement of computation power, datasets, and transformation of technologies. The performance of CNNs depends on their topology selection, selection of parameters, and training methods. In the early stages, the majority of architectural designs were mostly developed through manual processes, drawing inspiration from the problems posed by ImageNet. Nevertheless, the NP-hard problem [26] poses a significant challenge in manually determining the ideal parameters and their corresponding values in convolutional neural networks (CNNs). The development of a suitable architecture requires a substantial level of competence in the pertinent field, along with the application of an iterative approach for parameter selection. In recent years evolutionary algorithms have helped in designing efficient CNN architectures automatically. As shown in Figure 2.9, the evolution of architecture is divided into four categories. Experts designed the initial state-of-the-art architecture. However, after using evolutionary algorithms, its performance improved. As computation resources are affordable, deep learning-based models have become popular in multiple image classification and segmentation domains, such as medical, agriculture, defense, engineering, and many others.

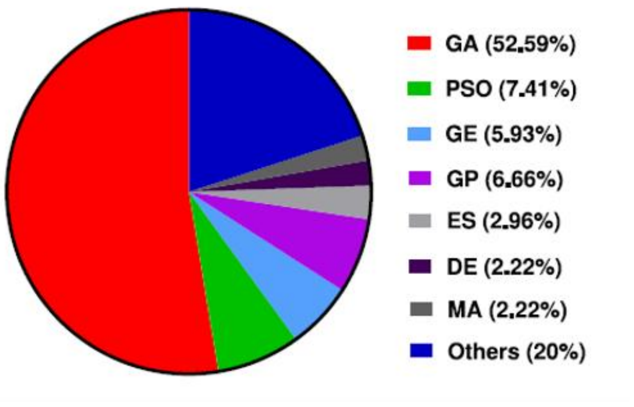


Figure 2.8 GA, particle swarm optimization (PSO), genetic evolution (GE), genetic programming (GP), evolution strategy (ES), differential evolution (DE), memetic algorithm (MA), and others that do not suit these categories[4].

The CNN model comprises topology selection, hyperparameter selection, variable initialization, and training. In topology selection, we need to identify basic building blocks such as pooling, convolutional, and fully connected and their interconnection. Hyperparameter selection consists of variables like learning rate, dimension, number of filters, pooling operations, and optimizer for the specific problem domain. In variable initialization, step initialization of learning parameters like filter values and weights in fully connected layers. In training filters, values are updated based on the feed-forward network with the multiplication of the learning rate. The hyperparameter selection in CNN is a complex task consisting of multiple feasible output solutions. Therefore, evolutionary algorithms can resolve the problem of multiple local optima and non-linear behavior of the parameters.

There are multiple evolutionary algorithm-based architectures are used for designing and training an optimal CNN architecture as shown in Figure 2.8 [4]. Evolutionary algorithms are population-based algorithm that works by selecting only a fit population for reproducing a new set of the population for the next iteration. The components of evolutionary algorithms based on their biological behavior must be designed by considering the problem domain.

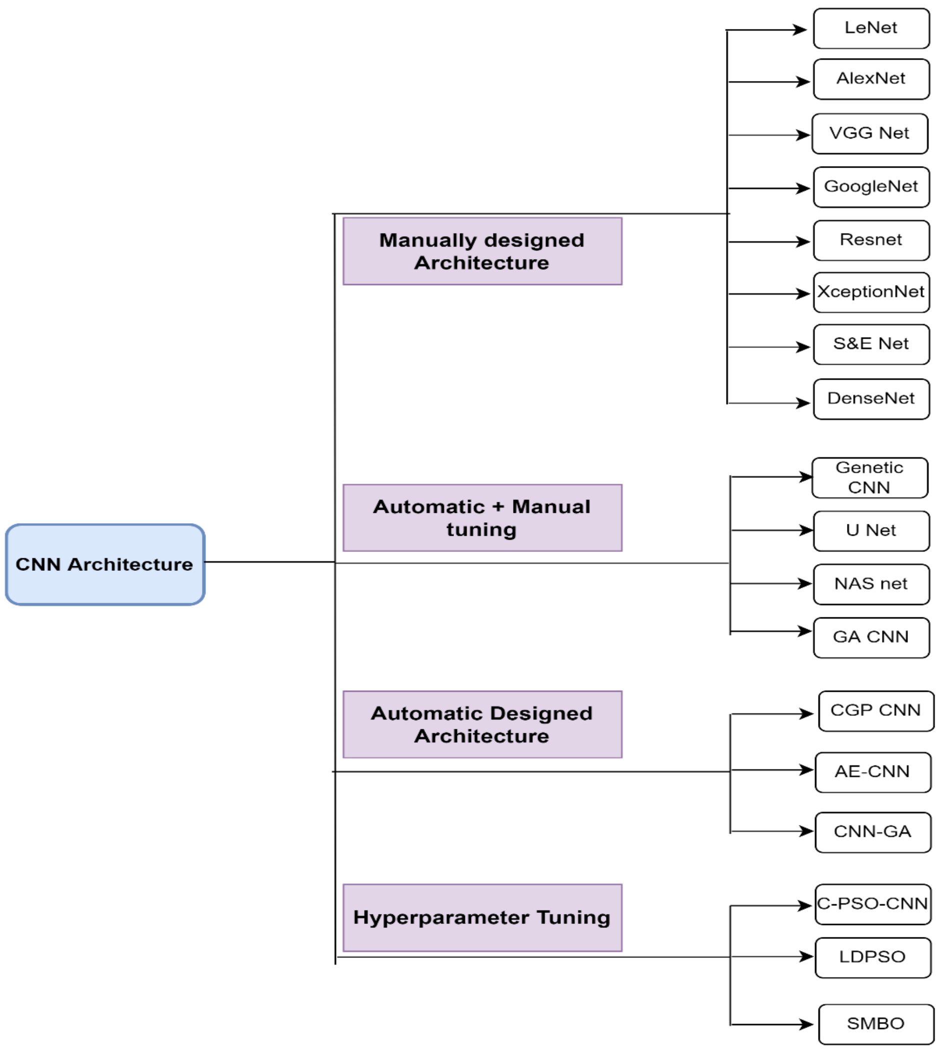


Figure 2.9 Evolution of deep CNN architecture

The effectiveness of the algorithm depends on the representation of components and their operators. For example, In GA, we have to redefine mutation and crossover operators based on the encoding scheme and fitness function. GA can be used for topology selection, training, and selecting hyperparameters, including filters, filter size, and the number of layers. The fitness function can be used to select the parameters which can be effectively applied to the CNN architecture. Figure 2.10 shows the methodology for using evolutionary algorithms in different steps of architecture selection. It starts with encoding representation that identifies the number of variables used to represent CNN architecture. That encoded architecture is passed along with the dataset to evolutionary algorithms. The evolutionary algorithm used the initialization method and passed decoded architecture with the dataset for training. After training, it is passed to the fitness function defined for the decision of whether the output is suitable for the next evolution process or discarded. After that, multiple operators are defined based on encoding to find suitable architecture. The process will continue till the maximum limit is reached. The selection of fitness function and maximum limit mostly regulate the computation power.

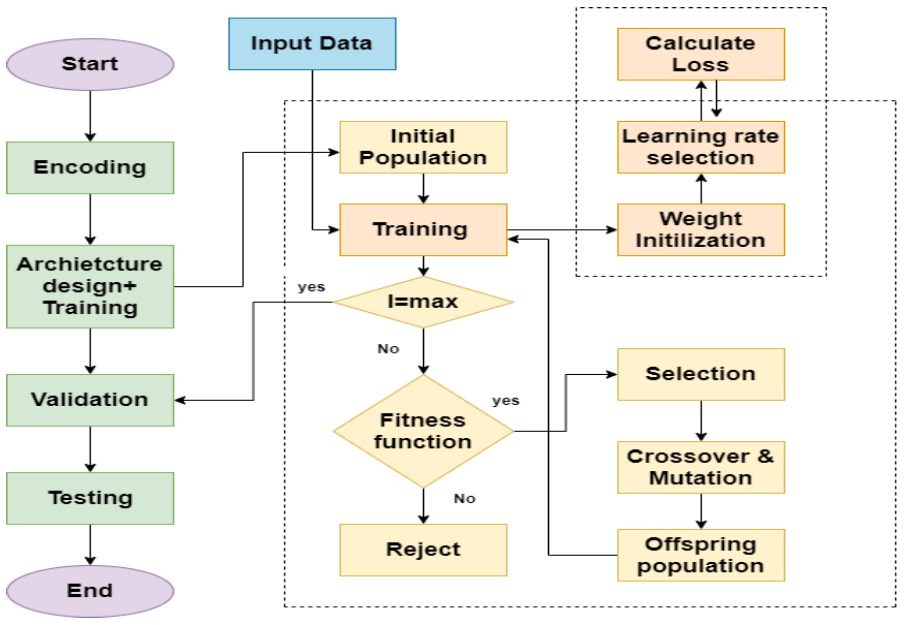


Figure 2.10 Methodology to represent CNN parameters selection using evolutionary algorithms and input data set

2.5.1 Encoding Scheme

To represent CNN architecture using evolutionary algorithms, we need an encoding method to represent the CNN architecture and pass in the evolutionary algorithm as input to test their effectiveness. Based on the literature, the encoding schemes are either fixed length or variable length. In the fixed-length encoding technique, the maximum depth of architecture is fixed. This technique is easy to implement, but it does not generalize the exploration. In Figure 2.11 example of fixed length encoding is explained. In Figure 2.11(a), two different chromosomes of 8 lengths represent CNN architecture where each chromosome is the combination of 3 units of convolutional layers, two units of pooling layers, and three units of fully connected layers present. In Figure 2.11(b) crossover operator is explained and in Figure 2.11(c) final output of the newly generated population is there. In dynamic encoding representation, we need to identify basic building blocks of CNN architecture, such as a convolutional layer, pooling layers, fully connected layers, and their interconnection. Each layer is represented by a few variables based on the applied encoding technique. Encoding of entire architecture is comprised of multiple small stages. Each stage is a different combination of basic building blocks.

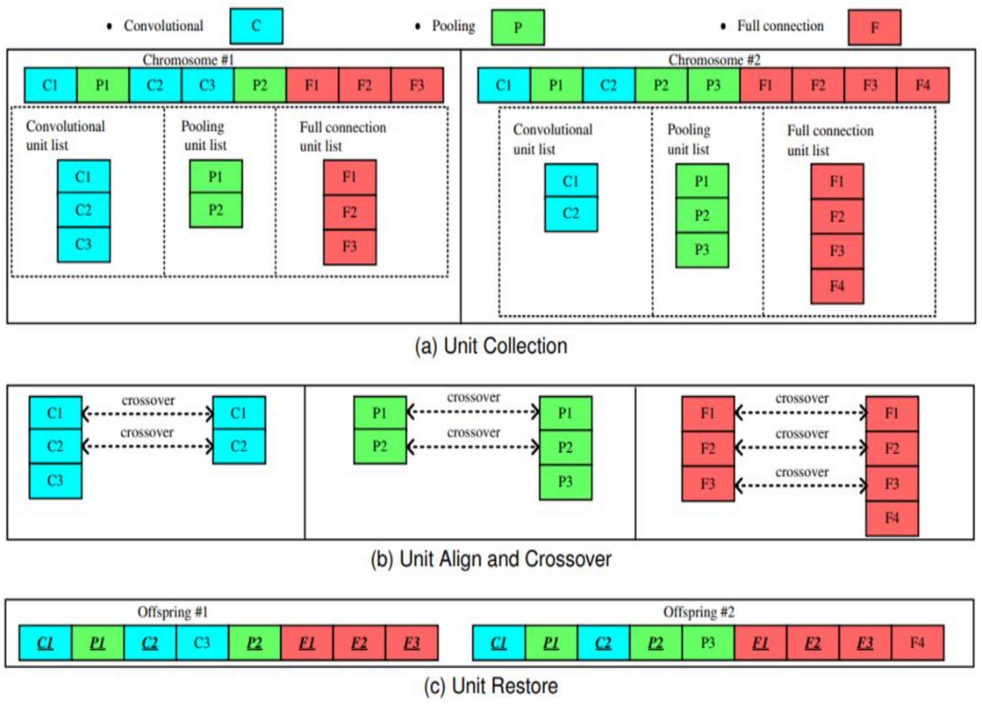


Figure 2.11 An example of CNN architecture representation using a fixed-length encoding scheme. The length of the chromosome is 8 and the mutation and crossover operator is defined to regenerate a new population of the same length [4].

In variable-length encoding representation depth of architecture is flexible. It is a more stretched encoded representation of CNN architecture. In variable-length representation, it is complex to define operators that make the population more diverse as compared to fixed-length representation. Figure 2.12 shows the different encoding representations. In Table 2.3, we compared different encoding representation techniques along with the methodology to decode and pass in evolutionary algorithms. Genetic NN used fixed-length binary encoding representation. It explains the interconnection between different layers of layers using 0 and 1 but is unable to justify the parameters within the layer. In CGP encoding 1st part represent the type of block and remaining two fields represent start and end node of architecture. In GACNN The architecture is represented by a concatenated block of a convolutional layer and a fully connected layer. The CNN-GA architecture is using variable length encoding, but due to huge parameters it computation cost increases with the depth of architecture. Also in AECNN three basic building blocks are used - DBU (Dense Block Unit), RU (ResNet Block), and PU (Pooling Unit). Due to fixed architecture its computation cost is less but in this representation modification of block is not possible that restrict to explore CNN architecture effectivly. We compared the number of chromosomes to represent the architecture with their complexity and diversity. If we use less number of chromosomes to represent, then the homological structure will be generated and stuck in local optima. If we use a different encoding for each encoding, then the overall computation cost will be very high. So based on the summary, it is observed that we need an encoding that can generate diverse architecture with demised computation cost.

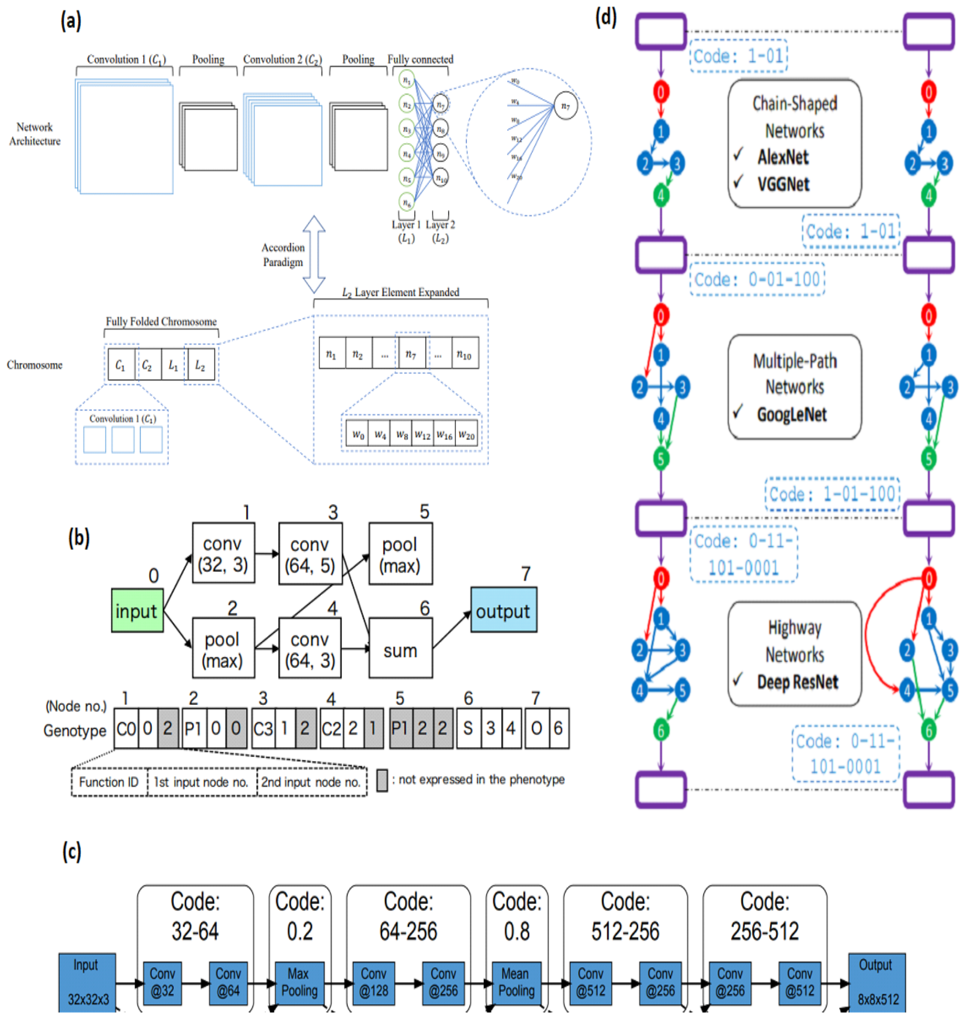


Figure 2.12 Block diagram of the encoding representation of CNN architecture in literatures reviewed; (a) GACNN [5], (b) CGP-CNN [6], (c) CNN-GA [1], (d) Genetic CNN [7]

Table 2.3 Methodology used to represent CNN architecture in existing encoding techniques

Model	**Encoding Representation**	**Methodology**
Genetic CNN, 2017 [7]	0-01-100	1. Existing architecture is divided into stages, where each stage is connected using binary encoding in increasing order. 2. The first '0' in the encoding representation indicates that stage 1 and 2 have no connection. The number '01' indicates that stage 3 is linked to stage 2 but not to stage 1. '100' indicates that stage 4 is directly connected to stage 1, but not to stages 2 or 3.

CGP-CNN, 2017 [6]	C3 | 1 | 2	1. Each node is divided into three blocks, with the first representing the operation field and the rest representing interconnection from the previous block. 2. The operations are defined as convolution, pooling, sum, and output. 3. Input layer defines the integer value of the previously connected input nodes. 4. Concatenated layers represent the entire architecture.
GACNN, 2019 [5]	C1 | C2 | L1 | L2	1. The architecture is represented by a concatenated block of a convolutional layer and a fully connected layer. 2. In the given encoding- c1, c2 are convolution layers with different filter sizes. A fully connected layer is represented by L1 and L2. 3. Pooling layer is not defined as it is fixed after each convolution block with max pool operation.
CNN-GA, 2020 [1]	32-64-0.2-64-256	1. CNN architecture is encoded into a string of decimal and fractional values. Two continuous decimal numbers represent the filter size of the convolutional layer and the fractional value represents the pooling layer. 2. If the fractional value is less than 0.5, then mean pool otherwise max pool operation is used.
AE-CNN [72]	1 | Type=1 5,128,64,20	1. Three basic building blocks are used - DBU (Dense Block Unit), RU (ResNet Block), and PU (Pooling Unit). 2. Variable length encoding is used. 3. Length is dynamic, which extends with a random selection of building blocks.
Sinha et al [14]	4 | 8 | 8 | 2	1. Four parameters are used- image size, filter size, number of filters, and architecture depth. 2. In this example, 9 bits are used to represent the entire architecture, 2 bits for each of the four possible image sizes, 3 bits for each of the eight different filter sizes, 3 bits for the number of filters, and 1 bit for each of the two possible depths.

2.5.2 Designing and training of CNN architecture

The complexity of CNN architecture plays an important role in learning complex features from training datasets. However, the difficulty of architecture increases with the depth of the architecture as well their interconnection. In addition to the depth, the number of parameters also increases. As a result, determining a sufficient depth with proper interconnection for any dataset is difficult, which is addressed by evolutionary algorithms. Genetic CNN [7] is used to explore the CNN structure automatically. In this algorithm, a fixed-length binary encoding scheme is used to represent CNN topology. The convolution layer and pooling layer are used as a basic building block of CNN. The Fully connected layer is not included in this representation. The entire architecture is divided into S stages, and in each stage, Ks convolution blocks are present in an ordered fashion, and only the lower number block is connected with, the higher number block. In training, architectures are trained from scratch, and the value of all hyperparameters is fixed throughout training. Then architectures are compared among each other, and the optimum architecture is identified. Now optimal architecture is compared with existing architectures state of the art with benchmark data sets like MNIST, CIFAR-10, and ILSVRC2012 [61]. This algorithm is not able to explore a lot of network structures because of the fixed-length encoding scheme.

In AE-CNN [80], GA is automatically used to design CNN architecture. This algorithm uses ResNet and DenseNet blocks as the basic building blocks. In the ResNet block, three convolution layers are connected using skip connections. In the dense block, four convolution blocks are present, where the individual block is connected to the rest of the blocks directly. The max pool and avg pool operations are chosen randomly in the pooling layer. Afterward, new crossover and mutation operators are designed according to the encoding scheme. This algorithm has shown significantly improved performance compared to the rest of the manually and automatically designed architectures. The state-of-the-art benchmark datasets such as CIFAR-10 and CIFAR-100 are used for comparing the architectures resulting in reduced training time with similar algorithms.

Cartesian genetic programming (CGP-CNN) [6] was presented to create CNN architecture automatically based on genetic programming [81]. This method uses a direct encoding scheme to represent CNN blocks and their connectivity. The benefit of this representation is that the depth of architecture is flexible and easy to implement skip connection also. It uses fixed block size to reduce the search space of architecture. In CGP encoding scheme, a direct acyclic graph is used to represent nodes and their interconnection. The mutation operator is used to replace unproductive nodes to evolve the architecture. This method uses parallel processing to reduce training expenses. This algorithm is tested on CIFAR-10 data set with a batch size of 128. It uses Softmax cross-entropy and Adam optimizer for computation efficiency.

To optimize the CNN architecture, it took about 14 days by CGP-CNN (ResNet). In the context of limited data, the author's approach is compared to VGG and ResNet. It has been noticed that VGG and ResNet models were found to be insufficient for handling tiny datasets, as they were primarily developed to perform optimally with large-scale datasets. Simultaneously, the CGP-CNN (ConvNet) and CGP-CNN (ResNet) models possess the capability to adjust their architectural configurations in accordance with the amount of the dataset. The optimization process for the design of CGP-CNN (ResNet) with limited data required a duration of five days. The proposed method can automatically make CNN architectures that are competitive and, in some cases, maybe better. This method has some shortcomings; it requires higher computational cost and time. Thus, the future work that can explore is to develop evolutionary algorithms that can reduce the computational cost. Another future work can be done to use this proposed method on different image datasets and tasks.

EvoCNN (Evolving Convolutional Neural Network) [82] is used to create an efficient algorithm for evolving an efficient architecture and automatically deciding the values of CNN weights. The authors specified many goals for building this system, such as developing a gene encoding technique and a weight initialization strategy. They also proposed a few operators to exceed loss function convergence. They proposed a fitness function with a minimized computing cost. They compare the proposed method and find that it outperforms the existing methods. The gene encoding strategy requires the basic layers of CNN to be encoded in one chromosome because the length of the gene was unknown before the architecture development. It is very suitable for the proposed method. As the performance of CNN is affected by the length of genes, EvoCNN has a better chance of reaching the best possible solution. To enhance comprehensibility, it is common practice to partition each chromosome into two distinct sections. The initial section encompasses the convolution layer and pooling layer, while the subsequent section encompasses the fully linked layer.

According to the length required, the chromosome is made, and the layers are attached. After making all possible combinations of layers, chromosomes are developed, and each goes through fitness evaluation. Parent solutions are found in the fitness evaluation test according to quantitative measures. This evaluation of each individual has two parts. The first is to train the CNN and the second is to calculate the fitness accuracy. During the preparation, first and foremost, each CNN is decoded dependent on the data addressed. Also, the loads concerning the convolutional layers and the completely associated layers depend on the Gaussian distribution by using the comparing mean and standard deduction encoded in the person.

Additionally, the CNN is prepared with the predetermined maximal ages with the clump information in the preparation set. The standard binary tournament selection is modified for this method. This method uses two sets of comparisons to select parent solutions. The comparison between mean values and the parameter numbers are the two comparison sets. If an individual is not selected through this method, the parent with a smaller standard deviation is selected. These comparisons are performed iteratively, and a set of selected parents is made. The offspring are created using genetic algorithms. At first, two random parent solutions are selected from the set; then, offspring are generated using the crossover operator. After that, the mutation operator is used, offspring are stored, and parents are deleted. This process continues till the parent set gets empty. Toward the finish of development, various individuals could have good mean values; however, various architectures and associations weight introduction values. There will be various decisions in choosing the "Best

Individual" in such a manner. For instance, if we are just worried about the best execution, we could disregard their design setups and think about just the classification accuracy. Something else, if we accentuate the more modest number of boundaries, relating choice could be made. When the "Best Individual" is affirmed, the comparing CNN is decoded dependent on the encoded design and association weight introduction data. Afterward, the decoded CNN will be profoundly prepared with a bigger number of ages by SGD for future use. Note that a ReLU non-linear layer is added to each convolutional layer and a complete association layer while interpreting the person to the comparing CNN.

This proposed method, EvoCNN, is compared with the best ten peer competitors, and the result shows that it outperforms all of them. Two state-of-the-art algorithms, GoogleNet and VGG16, have around 6.5% error rates, while EvoCNN achieved 0.83% to 5.47. EvoCNN performed equally in the mean performance to the two best algorithms, and EvoCNN has a smaller number of connection weights training epochs. This method promises much better performance because it evolves automatically in both the aspects of initial connection weights and the architecture. In further experiments, it is found that the EvoCNN method does not heavily rely on computing resources. It helps researchers without extensive expertise to create simple yet effective CNN models. EvoCNN was compared on nine benchmarks with 22 peer competitors. The results show that it outperforms all of them in classification performance. Using a smaller number of parameters, resulting in the least classification error rate, and using limited computational capacity and battery power shows that it can provide better choices to smart market devices willing to integrate CNN.

The algorithm CNN-GA [1] uses a variable-length encoding scheme to explore the depth of CNN topology. To efficiently search the search space and describe the encoding scheme, a modified crossover operator is also suggested. Skip connection is incorporated to deal with the data complexity problem and reduce the vanishing gradient. The author proposed an asynchronous parallel computation method to reduce the computation cost, which effectively utilizes computational resources. A novel catch component is introduced to reduce the fitness evolution costs by removing the overlapping computation. Each building block is designed in this encoding scheme using a skip layer and a pooling layer. The skip layer is a technique in which two convolutional layers are interconnected using a skip link. In the convolutional layer, the filter size of 3×3 and stride of 1×1 are set. In the pooling layer, the filter size is set to 2×2, and max/avg pooling operations are used.

In this technique, the fully connected layer is not used to reduce the search space for finding suitable architecture. Fitness values are calculated by training individual architecture using a given dataset. In this algorithm, cache is introduced to store the fitness value of evaluated individual architecture. This approach reduces the redundant calculation of individuals if it is already stored in the cache system. Based on the fitness value, vulnerable populations are eliminated. It is used to generate a new set of the fit population from the existing population using biological operators such as mutation and crossover. Mutation and crossover operators must be defined according to input data and fitness functions. This algorithm redesigned the crossover operator for unequal-length input strings, which improves architecture search. In this algorithm, multiple mutation operators are defined, such as adding or removing skip and pooling layers or randomly changing the building block parameter.

The mutation operator is employed to determine the optimal depth of an architecture. The efficacy of the CNN-GA algorithm is verified through experimentation on the CIFAR-10 and CIFAR-100 datasets. The results of this experiment indicate that the CNN-GA model exhibits superior performance in terms of accuracy and optimality compared to both manually-made CNNs and the hybrid automatic+manually constructed CNN. The CNN model developed by CNN-GA has a reduced parameter count in comparison to alternative CNN models. The CNN-GA approach requires fewer processing resources compared to the majority of automatic and automatic+manually built convolutional neural networks (CNNs).

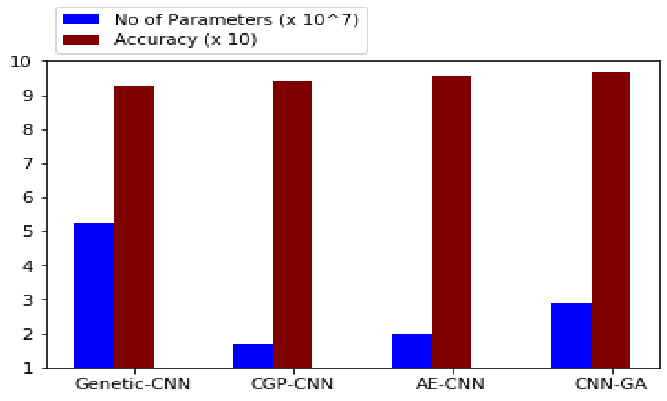


Figure 2.13 Comparison of evolutionary CNN architecture under CIFAR-10 dataset using genetic algorithm

We compared the recently developed evolutionary algorithms-based CNN architecture as shown in Figure 2.13. The CIFAR-10 dataset was employed by the author as a benchmark for evaluating the performance of a genetic algorithm With regards to precision and the quantity of variables. In this comparative analysis, it is observed that the CGP-CNN model has a lower parameter count, whereas the CNN-GA model demonstrates superior accuracy levels despite having similar parameter quantities.

In Table 2.4, we discuss the design of CNN architecture using evolutionary algorithms and compare it with the benchmark data set. This table compares most of the architecture designed based on genetic algorithms. A genetic algorithm is suitable to optimize non-linear parameters such as depth and width of architecture. The total parameters are proportional to depth, width, and interconnection. The complexity of the model and the number of parameters also depend on dataset complexity. So to implement CNN architecture using evolutionary algorithms, different encoding techniques are discussed in this table, performance is compared, and relevant gaps are identified. The performance is compared based on accuracy, the number of parameters and training cost, and processing power.

Table 2.4 Comparisons of recent evolutionary architectures based on CIFAR-10, CIFAR-100, and MNIST datasets.

Reference	Architecture	Data set	Accuracy/ Error	GPU	GPU Days	Parameters (in millions)	#epochs	Relevant review findings
[7]	Genetic CNN	MNIST	99.66	Titan -X	-	-	50	1. Fixed-length binary encoding scheme is used. 2. Existing architecture is represented using the different connections of convolutional layers. 3. All convolutional layers have the same number of filters. 4. Encoded architecture is initialized using the Bernoulli distribution.
		CIFAR-10	77.06		17	-	50	
		ILSVRC	27.87 (error)		-		50	
[82]	EVO CNN	Fashion benchmark dataset	5.47	-	-	6.68	100	1. Use of a smaller dataset; the model is not tested on a large dataset. 2. Three basic building blocks like convolution layer, pooling layer, and fully connected layers are used. 3. Mean and standard deviation of weight values are used for initialization.
[1]		CIFAR 100	20.85		36	5.4	-	

	CNN-GA	CIFAR 10	96.78	Nvidia Geforce-GTX 1080 Ti	35	2.9	-	1. Initialization is done using normal distribution. 2. Fixed length encoding scheme is used. 3. Semi-folded architecture reduces the number of comparisons in chromosome labels. 4. Pooling is not part of the proposed encoding representation. 5. Not able to justify skip connection in this encoding.
		CIFAR 100	79.47		40	4.1	-	
[6]	CGP-CNN	CIFAR 10	94.02	Nvidia Geforce-GTX 1080	27	1.68	-	1. This representation is unsuitable for representing complex architectures because it only uses two fields for concatenation. 2. Exploration is limited due to the limited number of pooling and convolution layers. 3. The number of nodes ranges from 10 to 50.
[80]	AE-CNN	CIFAR 10	4.3	-	27	2	-	1. ResNet and denseNet blocks are considered basic building blocks, or a combination of these blocks is considered a basic block. 2. Genetic algorithm is used to optimize the architecture.

2.5.3 Hyperparameter optimization

Selection and initialization of hyper-parameters are critical in finding optimal structures. The utilization of this method contributes to the mitigation of training expenses and facilitates the attainment of the global optima at an accelerated rate. The quantity of hyper-parameters in intricate structures is substantial, therefore, it is difficult to initialize them manually. For instance, the AlexNet model has 27 hyperparameters, VGG Net has 57 parameters, GoogleNet has 78 parameters, ResNet-52 has 150 parameters, and DenseNet has 376 hyperparameters. Therefore, it is almost impossible to identify the suitable combination of parameters manually. The selection of optimum parameters is an NP-hard problem, and researchers are examining a metaheuristic solution for such problems. Many genetic algorithms, swarm intelligence, and evolutionary algorithms are used to solve this problem.

Swarm intelligence is used to solve such nonlinear, multidimensional hyperparameter selection problems. Talathi (2019) introduced a novel approach known as sequential model-based optimization (SMBO) for the purpose of selecting hyperparameters in deep convolutional neural networks (CNNs) [19]. In this methodology, the author has taken into account a limited number of parameters across various layers. These parameters include the quantity of convolutional layers, the quantity of filters within each layer, the size and stride of the filters, the inclusion of normalization layers, the configuration of pooling layers (including size, stride, and type of pooling, whether max or average), the number of hidden layers, the number of nodes within each hidden layer, and the specified dropout value. In this model p- ReLU (parametric rectified nonlinear unit) activation function is used. The overall model is tested on the CIFAR-10 dataset, and a 6.9% mean error is obtained. Toshi et al. proposed a PSO-based hyperparameter selection algorithm that improved classification accuracy [14]. In this analysis, the author chooses a few input parameters such as image size, filter size, number of filters, and depth of the architecture. Each parameter has a range of discrete values, out of which a suitable one is considered. This model uses four possible image sizes, eight different filter sizes, and two different architectures consisting of 11 and 13 layers. This model is trained using CIFAR-10 and RSVD datasets. RSVD is the author's self-created dataset consisting of 653 images and seven classes of road, green grass, brown grass, soil, sky, leaf, and tree stem. PSO is applied to identify the most suitable architecture providing an accuracy of 88.3% in RSVD and 81.47 in the CIFAR-10 dataset.

Another PSO-based algorithm known as canonical PSO (cPSO-CNN) was introduced for optimizing CNN parameters [83]. This used compound normal distribution to improve the exploration capability of CNN. It uses a new prediction model to reduce the cost of the fitness function. The cPSO-CNN architecture was compared with existing architecture such as AlexNet with the CIFAR-10 dataset, and performance was improved.

Table 2.5 Performance comparison of CNN hyperparameter optimization using evolutionary algorithms

Reference	EA Method	CNN Hyperparameters	EA Parameters				Accuracy /Error	Parameters (in millions)
			Population size	Iteration	Fitness function	Data Set		
[19]	SMBO	1. Number of layers 2. Number of filters per layer 3. Filter size 4. Stride 5. Types of pooling 6. Dropout value 7. Activation function	NA	NA	CNN	CIFAR -10	6.90	3.4M
[14]	PSO	1. Image size 2. Filter size	4	10	CNN	CIFAR -10	81.47	NA

		3. Number of filters 4. Depth of architecture	10	40	CNN	RSVD	88.27	NA
[83]	cPSO-CNN	1. Convolutional layer 2. Pooling layer 3. FC layer 4. Filter size 5. Number of filters 6. Stride 7. Padding	NA	195	Fast Fitness evaluation	CIFAR-10	8.67	NA
[84]	PSO	1. Number of filters 2. Size of filter 3. Activation function 4. Fully connected layer 5. Batch size 6. Optimizer	10	10	NA	CIFAR-10	69.37	NA
						MNIST	98.95	NA
[85]	GA	1. Number of filters 2. Size of filters 3. Depth of architecture	8	NA	NA	Stop Sign Image	98.94	NA

Some parameters are decided during the design phase, such as depth of architecture, layers and their interconnection, and learning rate. However, the value of a few parameters is decided during the training phase, like the number of filters, size of filters, batch size, learning rate, and stride & padding. In Table 2.5, we include the research paper that compares the architecture performance based on parameter selection. So selection of optimum parameters and their value is based on different algorithms like GA and PSO.

Evolutionary algorithms benefit architecture selection, parameter tuning, and architecture training in deep neural networks such as ANN, CNN, and RNN architecture designs. The model's performance is evaluated based on accuracy, the number of training parameters, hardware requirements, scalability of data set, convergence speed, and number of GPUs involved. The relationship between the number of parameters required for constructing an architecture and the accuracy is indefinite in the existing literature. Identifying which architecture is better is indefensible as it involves many parameters. Parameter selection and model training is difficult in all deep neural network models. Architecture's performance is mostly evaluated based on accuracy and the number of parameters. The number of parameters is directly proportional to the training time of the architecture as well as the memory required for storage. Figure 2.14 shows the comparison of evolved architecture that shows the accuracy of an evolutionary-based model is equivalent to manual design architecture with a huge margin in the number of parameters. That shows the effectiveness of the evolutionary algorithm in deep architecture selection.

Although EA performs considerably well in solving the above problem but involves a few challenges that need to be discussed, EA-based algorithms can generate numerous CNN architectures. We train all architectures and compare their performance while selecting the most suitable architecture. The training cost of a deep CNN architecture requires a good amount of computational power, which gets multiplied by the number of architecture designs. It is a primary challenge to design an efficient evolution method that can optimize the training cost of unique architecture. We require efficient operators that can generalize and effectively utilize the search space. A novel method that can reduce the number of parameters and optimize the training costs quickly is required.

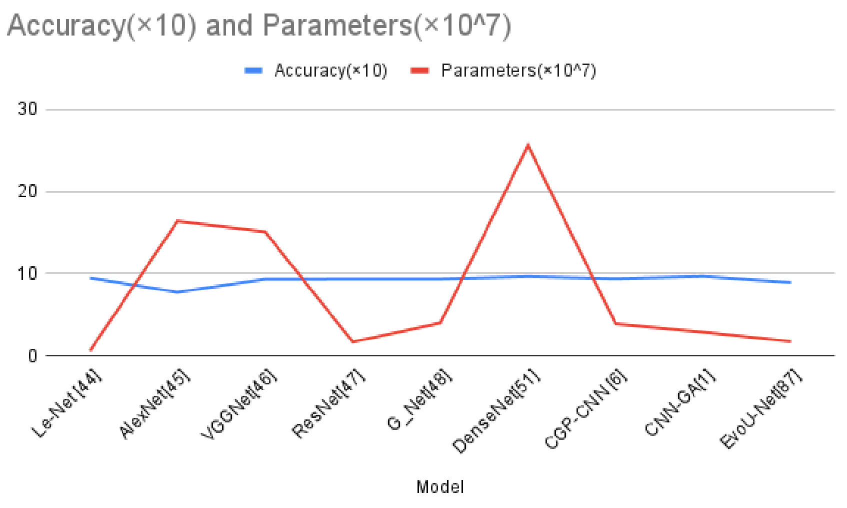


Figure 2.14 Comparisons of evolved architectures under CIFAR-10 dataset

The performance of EA is better than the random selection of parameters, but it does not guarantee the optimal parameter selection. The convergence rate in EA is faster in initial iterations and becomes slower when it is closer to the solution. Initialization of parameters is an additional problem that may cause the outcome to be stuck in local minima. We also use a suitable activation function to increase nonlinearity. Determining a suitable encoding task for a CNN architecture is challenging. The encoding representation of a CNN architecture is a demanding task because it supports exploring several CNN architectures. There are two major classifications of encoding methods, specifically fixed-length encoding schemes and variable-length encoding schemes. The adoption of a fixed-length encoding approach in the architecture guarantees a consistent depth, hence enabling the smooth integration of the Convolutional Neural Network (CNN) and its corresponding operator. The utilization of a variable-length encoding strategy facilitates the exploration of numerous convolutional neural network (CNN) architectures. However, its implementation is challenging due to the significant computational expenses it incurs.

The CNN architecture designed and trained for one dataset may not be efficient for another type of dataset. The performance of CNN architecture relies on the size of the data set, image resolution, quality of images, and data distribution.

The existing CNN models, as discussed, are incompetent to adopt across multiple domains as it requires a variety of datasets in large quantities for better performance, especially in the medical domain where the dataset is scarce. Additionally, it is required to have ample knowledge in the problem domain and architecture design. The automatic designing of CNN architectures is required to resolve the issue by constructing the architecture using EA algorithms. However, it is unaffordable in a medium-scale educational institute and small-scale originations due to the high computational cost involved in designing the architectures.

Table 2.6 Comparison of manual and automatic architectures based on different evolutionary algorithms

Ref	**Model name (CNN)**	**Data Set**	**Accuracy (%)**	**Parameters (in millions)**	**Training Cost**	**Approach**
[44]	Le-Net	MNIST	95.00	0.60	-	Manual
[45]	AlexNet	CIFAR-10	77.50	16.4	-	Manual
[46]	VGGNet	CIFAR-10	93.34	15.1	-	Manual
[47]	ResNet-110	CIFAR-10	93.57	1.7	-	Manual
[48]	GoogLeNet	CIFAR-10	93.64	4	-	Manual
[51]	DenseNet	CIFAR-10	96.54	25.6	-	Manual
[85]	NAS	CIFAR-10	93.99	2.5	22,400 GPU days	RNN
[86]	NAS-Net	CIFAR-10	96.27	2.6	2000 GPU days	RNN
[28]	Block-QNN-S	CIFAR-10	95.62	6.1	90 GPU days	RNN
[83]	LDWPSO	CIFAR-10	69.37	-	10 Epoch	PSO

[80]	cPSO-CNN	CIFAR-10	81.47	-	-	PSO
[1]	CNN-GA	CIFAR-10	96.78	2.9	35 GPU days	Genetic Algorithm
[87]	EvoU-Net	PROMISE-12	89.30	1.76	20 Epoch	Genetic Algorithm
[88]	FAST-CNN	CIFAR-10	94.70	-	14 GPU days	Genetic Algorithm
[89]	Tabu_Genetic Algorithm	MNIST	99.38	-	22 Epoch	Genetic Algorithm
[6]	CGP-CNN	CIFAR-10	93.95	3.9	31 GPU days	Genetic Algorithm
[7]	Genetic CNN	CIFAR-10	92.90	-	20 GPU days	Genetic Algorithm

This article focuses on comparing architecture selection and hyperparameter tuning, as shown in Table 2.6. However, architecture training is one area where we need to check the application of EA algorithms because there may be a chance of being stuck in local maxima in a derivative-based training mechanism. This problem can be solved with the help of EA. Additionally, most of the CNN architecture is trained for large datasets that are inadequate for smaller datasets, such as medical domains like bone fracture classification and segmentation [25], and some real-world problems like medicinal leave classification [90], where we have limited datasets available. Other areas like RNN, NAS [91], encoder, time series prediction, and signal processing can also use EA-based algorithms to check the feasibility.

The neural network architecture is used in multiple domains to solve complex problems and predictions. It uses to develop virtual assistants, chatbots, healthcare domain, entertainment, fake news detection, image recognition, image classifications, segmentation, robotics, and many more. It can also be used to design RNN based model for time series data analysis. It can be used in domains such as healthcare or agriculture, where researchers are not experts in

CNN designing. They can use an evolutionary algorithm to design automatically from scratch without any intervention. It can also use in hyperparameter tuning of deep neural networks.

2.6 Conclusion

The CNN has shown immense success in computer vision and image classification tasks, which extract image content higher representations. This article provides a comprehensive survey of the evolution of CNN architectures using various evolution algorithms. We have compared EA-based architectures with the existing CNN architectures created manually on different aspects such as accuracy, depth, number of parameters, and computation cost. In this survey, we have examined different EA algorithms like GA and PSO and their applications to optimize the parameters of convolutional neural networks. We have compared EA from four aspects: encoding technique, population initialization, EA operators like mutation and crossover, and fitness function. Table 2.6. summarizes the performance of various CNN designs in benchmark data sets. We have analyzed different manually-designed state-of-the-art CNN architectures in the image classification dataset and addressed the issues and challenges in the existing techniques that can help in future research work. It is challenging to manually select and initialize the parameters because of their large size. The recent study areas, EA-based algorithms such as GA and PSO have been employed to handle the problem with satisfactory results. However, a few issues and challenges, including proper encoding technique, weight initialization, and computation resources, still exist. Additionally, different EA-based operators like mutation and crossover need to be redefined to cover maximum search space efficiently. We must define a good fitness function that can optimize the performance and reduce the number of comparisons.

CHAPTER -3

ENCODING REPRESENTATION OF CNN ARCHITETCURE

3.1 Introduction

To design CNN architecture using an evolutionary algorithm first we need to define encoding representation to represent CNN genotype and phenotype. The literature analysis reveals that encoding schemes can be roughly categorized into two types: This discussion pertains to two types of encoding methods: fixed-length encoding schemes and variable-length encoding schemes. The implementation of a fixed-length encoding method requires the first determination of the architectural depth. One of the benefits of utilizing fixed-length encoding representation is its ease of implementation on pre-existing architecture, making it well-suited for defining mutation and crossover operators. Genetic CNN [7], and Evo-CNN [82] are used fixed-length encoding to define CNN architecture. In the variable-length encoding scheme depth of architecture is not restricted which helps to explore the architecture more generously. But it is quite a complex task to define the genetic operation to mutate the architecture. Architecture such as CNN-GA [1], CGP-CNN [6], and AE-CNN [80] is used variable-length encoding representation. The effectiveness of encoding representation is evaluated based on computation cost, accuracy, number of parameters, and adaptability. If encoding schemes are represented with only a few parameters then It will restrict the exploration of the architecture. If we represent individual training parameters as one unit then the number of possible chromosomes is too huge which will increase computation cost.

In Table 3.1, we compared the effectiveness of different existing encoding schemes in terms of the number of parameters and training time in GPU days with different datasets. After discussing relevant gaps in different encoding schemes, a novel adaptive coding model is proposed. Figure 3.1 depicts a comparison of different encoding schemes based on architecture using the CIFAR-10 data set using genetic algorithms. The proposed scheme is adaptive & versatile in nature. A simple representation offers a better understanding of the complex network. Adaptive behavior scales up the application domain of the proposed scheme.

Table 3.1 Comparative analysis of existing encoding techniques

Model	**Dataset**	**Parameters**	**GPU days**	**GPU**	**Relevant findings**
Genetic CNN, 2017 [7]	CIFAR-10	0.52M	17	Titan-X	1. Fixed-length binary encoding scheme is used. 2. This encoding is used to represent an existing model. 3. All convolutional layers use the same number of filters and filter size.
	ILSVRC 2012		20		
CGP-CNN, 2017 [6]	CIFAR-100	0.83 M	14	Nvidia Geforce-GTX 1080	1. This representation is suitable to represent basic architecture as only two fields are used for concatenation. 2. The limited number of pooling and convolution layer make it restricted to explore. 3. Number of nodes is between 10 and 50.
CNN-GA, 2020 [1]	CIFAR-10	2.9M	35	Nvidia Geforce-GTX 1080 Ti	1. Variable length encoding is used. 2. Fixed filter dimension 3X3 and stride value 2 is used in the convolutional layer. 3. Fully connected layers are not part of the representation.
	CIFAR-100	4.1M	40		

3.2 Linear encoding scheme

In this algorithm, we pass input datasets, and after a sequence of evolution, the framework automatically evolves to a suitable CNN architecture. A random population is initialized using a predetermined encoding and population size throughout evolution. Figure 3.2 depicts an example of the variable length encoding system employed in the proposed study. This representation uses a 32×32 dimension colour image as the input to the convolutional layer. The number of filters in a convolutional layer is randomly selected using population initialization methods. The dimension of a filter is fixed to 3×3, and a stride of 1×1 is used to make it homogenous and reduce the computational cost.

In the pooling layer, the algorithm automatically selects avg pool or max pool operation with equal probability having kernel size of 2×2 and stride 2×2. The concatenated string of different layers represents the encoded representation of CNN architecture, as shown in Figure 3.2. In pooling layer representation, it shows with the pooling operation either min, max, or average pooling along with kernel size 2×2 and stride 2×2. The concatenated string represents the encoded representation of CNN architecture.

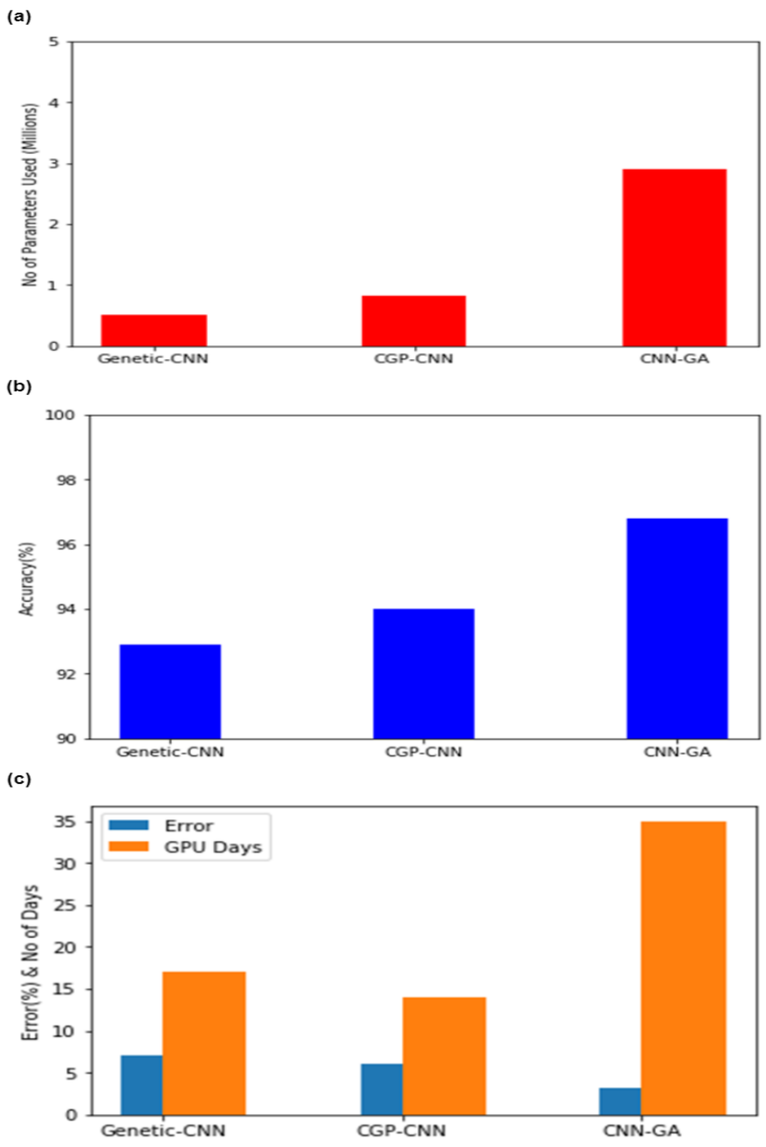


Figure 3.1 Comparison of various encoding schemes under training in CIFAR-10 dataset using genetic algorithm; (a) Accuracy achieved, (b) Number of parameters used, (c) Error rate and training cost

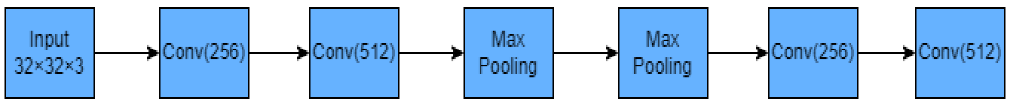


Figure 3.2 Decoded architecture of encoding representation “256-512-max-max-512-256”

The hyperparameters are manually chosen using the existing state-of-the-art model. The fitness of each individual, which represents a distinct convolutional neural network (CNN) architecture, is evaluated during the process of evolution using the given dataset. In the succeeding generation, parental individuals are chosen according on their fitness, and new offspring are produced using genetic operators such as crossover and mutation. The recently generated population is merged with the preexisting population to form a novel roster of progeny. The process of development continues until the counter surpasses the maximum generation, with the counter being incremented by one. Most existing frameworks are developed using fixed maximal generation, which could restrict resource management. This paper employed adaptive exit conditions that terminate automatically when the convergence rate is slow or near zero.

3.3 Hybrid encoding scheme

This section details the proposed encoding scheme to represent CNN topology. The proposed scheme employs a variable-length encoding scheme that represents the depth as well as the width of the architecture. The scheme comprises four basic building blocks as shown in Figure 3.3. A few bit strings represent each building block and concatenated structure will represent the complete CNN architecture. In this method, we define some basic building blocks as the genesis block, transit block, agile block, and fully connected block. A few bit strings represent each building block and concatenated structure will represent the complete CNN architecture. The genesis block is the combination of one convolutional layer followed by one pooling layer, which is starting block in the architecture. The complexity of an architecture is increased by different combinations of agile and transit layers, and the fully connected layer is used at the end.

The agile block is a concatenated convolutional block with a fixed number of filters and the dimension of filters is also fixed. To represent the agile block using an encoding scheme we used five concatenated parameters, the first one represents operation, the second one represents the size of filters, the third one represents the number of filters, the fourth parameter represents the depth and the last one represents the interconnection of a different convolutional layer using binary encoding. The size of filters and number of filters are randomly initialized with pool of data created using literature study. The depth of architecture also generated randomly in the range of [3-5]. Their interconnection is represented by binary encoding proposed in genetic cnn.

In the transit blocks we define operational block followed by batch normalization and then pooling operation in fractional part. Value of pooling operation is define in range of [0-1]. If value is less then 0.5 means max pool operation is used else mean pool opearation is used. Batch normalization is 1x1 convolution operation is used to scale down filters dimensional to concatenate different size of input features. In fully connected layers first parts represent operational value, after that k1 represents the number of results in flattern operation and k2 represents number of output classes.

The main advantage of the proposed encoding scheme is that it can represent architecture with a combination of two different layers. It makes the representation simple and one can increase the depth of architecture easily. Also, due to fewer parameters, one can define different evolutionary operations like mutation and crossover efficiently. The scheme also supports increasing the complexity within the block. In the agile block, it can generate filer size and depth randomly and thereby increases complexity. The proposed scheme supports a hybrid encoding scheme that utilizes binary as well as decimal representation. The encoding scheme offers the maximum choice of exploration in depth and width as well as faster optimization. We pass our initialized encoding method in evolutionary algorithms to optimize for better architecture. The maximum number of iterations is fixed at 50 as limited computation power is available.

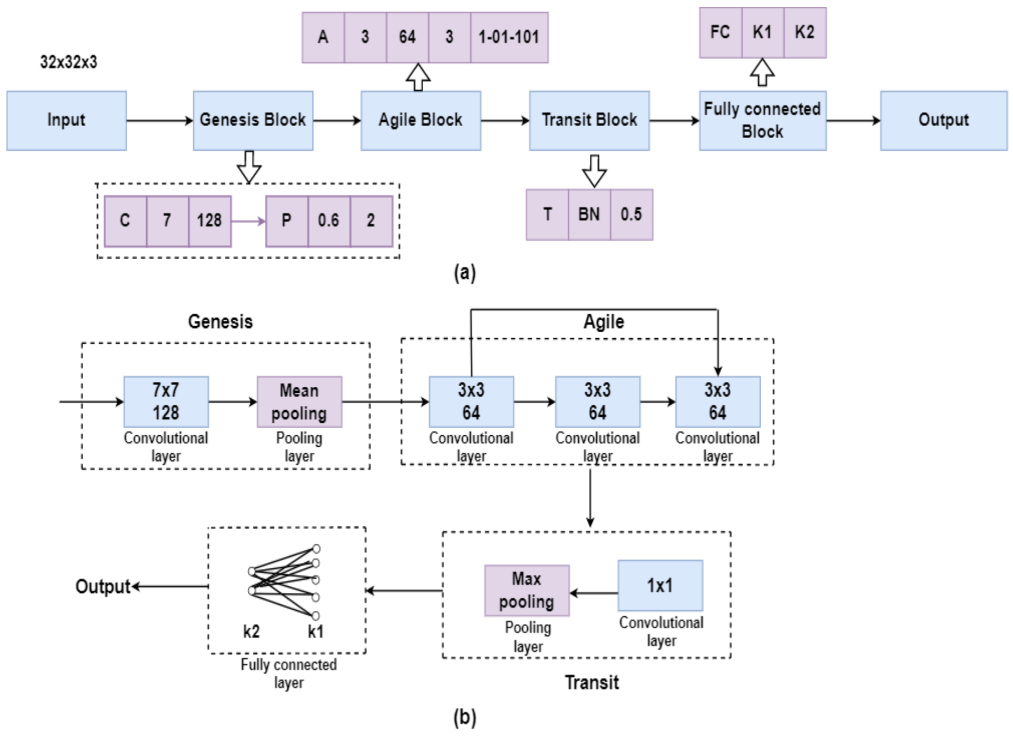


Figure 3.3 Hybrid encoding (a) Block diagram of the proposed encoding representation (b) CNN architecture of corresponding proposed encoding scheme [92]

The performance of architecture depends on topology design as well as hyperparameter selection [93]. In this paper, we fixed the value of learning parameters based on the previous literature such as we chose filter dimensions 1x1, 3x3, and 5x5, and a number of filters 64, 128, and 256 in a convolutional layer and agile layer [94]. We propose depth in range 3 to 7 in the agile layer to maintain simplicity. For the weight initialization of a fully connected layer, we suggest using transfer learning in the CIFAR-10 benchmark dataset for evaluation of performance as the limited computational power is available else CIFAR-100 or any complex dataset can be used. In the fitness function, we propose to use only 10% of the dataset to select the next population which makes it faster, and after the selection of the top architecture, we trained for the complete dataset. Input image size is proposed as 32x32, with 128 batch size and the learning rate is chosen as 0.1 for homogeneous data size.

A novel encoding scheme for representing CNN architecture is proposed which can effectively be used to represent a complex architecture with a variable number of parameters. The study also presents a decisive comparison among various existing encoding schemes that can help the researchers in choosing the best suitable method for their application-specific projects. The comparative analysis highlights the merits and demerits of existing schemes through multiple parameters like accuracy and computational power. The authors also represented a depth analysis based on the number of parameters used to represent input chromosomes, their initialization methods, operators used to find different combinations, and fitness function to stop the searching methods.

3.4 Conclusion

The study proposed a novel encoding method that is used to represent complex CNN architecture. In this encoding, we can represent existing architecture as well as generate new architecture using an available dataset. It covers both the depth and width of architecture that reduces the number of parameters and helps to identify comparable architecture in significant improvement of computation power with comparable accuracy. This encoding scheme is used to pass evolutionary algorithms to design new architecture automatically using different datasets. We can use evolutionary algorithms for hyperparameter tuning and use this encoding representation in the future.

CHAPTER -4

AN EVOLUTIONARY FRAMEWORK FOR DESIGNING ADAPTIVE CONVOLUTIONAL NEURAL NETWORK

4.1 Introduction

This part provides a description of the structure of the proposed algorithm in subsection 4.2, followed by an analysis of its crucial points population initialization in subsections 4.2.1, fitness function in subsection 4.2.2, and offspring generation in subsection 4.2.3. In order to facilitate comprehension of the algorithm under consideration, we shall provide a comprehensive account of the details associated with each notable stage, with an evaluation of specific architectural-level designs.

4.2 Algorithm overview

The proposed algorithm's framework is shown in Algorithm 1, and the flow chart of the proposed framework is depicted in Figure 4.1.

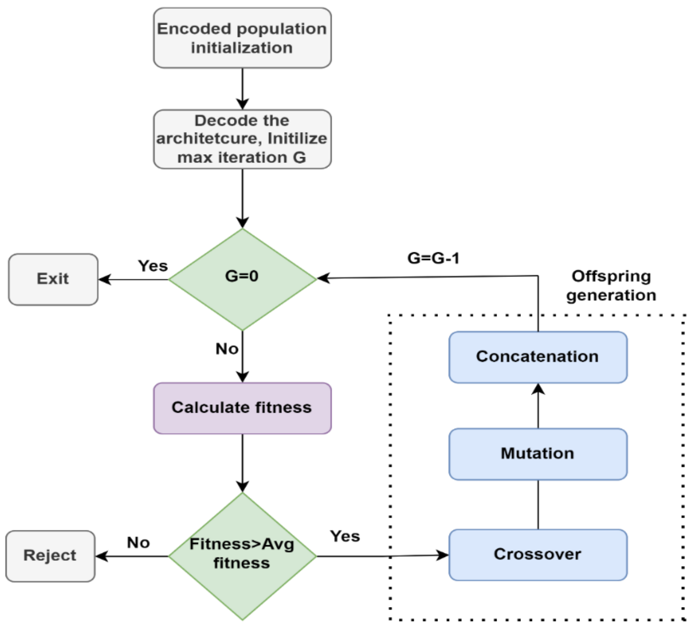


Figure 4.1 Flow chart of evolutionary algorithms

In this algorithm, we pass input datasets, and after a sequence of evolution, the framework automatically evolves to a suitable CNN architecture. A random population is initialized using a predetermined encoding and population size throughout evolution. Figure 3.2 depicts an example of the variable length encoding system employed in the proposed study.

Algorithm 1 Framework of the proposed algorithm using EA

```
Input: A dataset of a set of CNN architectures represented by the
variable length encoding technique.
Output: Identifies the best CNN architecture.
  1. Propose an encoding scheme to represent CNN architecture.
  2. Initialize the population of N CNN architectures with the help of
     the proposed encoding method. Initialize max iteration G, the
     number of epochs for the fitness function, and the input dataset.
  3. Initialize the hyperparameter kernel size, loss function, learning
     rate, and stride size.
  4. While (G>0)
       4.1 Calculate the fitness of each architecture.
       4.2 Select N/2 best architectures for reproduction using GA
           operators.
       4.3 Apply crossover and mutation operators to generate new
           offspring.
       4.4 Concatenate the new population with the existing best
           population to create a new pool of N architectures.
  5. G ← G-1
  6. End
  7. Return the best CNN architecture.
```

4.2.1 Population Initialization

The main components of a Convolutional Neural Network (CNN) consist of convolutional layers, pooling layers, and occasionally fully connected layers. The performance of the CNN is significantly influenced by its parameters, which are depending upon the depth and width of the network's connections. The fully connected layer is omitted in this encoding due to its computational inefficiency caused by the large number of parameters.

Algorithm 2 Population Initialization

```
Input: The number of initial population N.
Output: The list of N initialized architecture using encoding representation.
1. P 🡐 Ø
2. While |P| < N
3. Choose random integer D as depth.
4. Generate a convolutional layer with the number of filters between $[2^5 - 2^9]$ and filter size is 3×3.
5. While (D>0)
   5.1. Choose a random number between (0-1)
   5.2. If number < 0.5
        5.2.1. Generate a convolutional layer with the number of filters are
               between $[2^5 - 2^9]$ and a filter size is 3×3.
   5.3. Else
        5.3.1. Choose between max pool and avg pool randomly.
        5.3.2. Concatenate the selected layer with the existing architecture
               Pi.
   5.4. D--;
6. P=P U Pi
7. End
8. Return P.
```

Initially, the number of population and the depth of each population is selected randomly. In the selected population, the first layer is fixed as a convolutional layer; then, convolutional and pooling layers are determined randomly with equal probability. The convolutional layer's filter count is randomly chosen in the range of $[2^5 - 2^9]$. All the selected population is organized in a list to evaluate the fitness value after initialization. The filter size and pooling operation range are selected manually based on a few standard architectures. The algorithm for population initialization is mentioned in Algorithm 2.

4.2.2 Fitness function

Algorithm 3 evaluates the fitness of all input populations using a given dataset. An individual's CNN is initially decoded using a predetermined set of hyperparameter parameters. CNN decoding is trained with training data, and accuracy is used to determine fitness. Because the training of CNN is a time-taking task, we used half of the dataset for initial training to make it efficient. After training the population, half of the population is eliminated based on fitness score. The best population is chosen for reproduction in the following offspring generation. If the model is showing good training accuracy, but validation accuracy is not increasing in respective of training in a few successive epochs, then architecture may suffer from overfitting [95]. We can eliminate the overfitted model to reduce the computation cost in the early stages.

Algorithm 3 Fitness function

```
Input: The selected population list of CNN architecture, input dataset, range
of hyperparameters, optimizer, loss function, epoch, train data, and test data.
Output: Best CNN architecture with fitness value
  1. Divide the dataset into train and test data.
  2. $F_{best} \leftarrow 0$
  3. For each population $P_i$ in population pool P do:
  4. Decode the architecture and calculate fitness accuracy using half of
     the population using backpropagation methods.
  5. Eliminate the architecture based on overfitting.
  6. Choose P best population, train using the complete dataset, and
     calculate fitness value F for each.
     6.1 If  $F > F_{best}$
     6.2 $F_{best} = F$
     6.3 End
  7. End
```

4.2.3 Offspring generation

Algorithm 4, consisting of two parts, illustrates the specifics of producing the offspring. Crossover is the first, and mutation is the second. Specifically, two parents are selected based on which of two randomly selected individuals is more suitable. We build a new set of populations with equal probability by utilising mutation and crossover processes. In a crossover operation, each parent is arbitrarily divided into two pieces, and the two pieces from each parent are exchanged to generate two offspring. We have chosen crossover probability 0.8 and mutation probability 0.2. Mutation operation helps define the architecture's exact depth, whereas the crossover operation increases the convergence rate. Both operations must be compatible with the encoding scheme. Newly generated offspring will be combined with the previous best architecture to create a new population pool.

Algorithm 4 Offspring generation

```
Input: Input population list P, with its fitness value, mutation, and crossover operation with their probability value.
Output: Newly generated population list $Q$.
1. Q ← Ø
2. While | $Q_t$|<|$P$ | do
    2.1. p1,p2 ← randomly select two population values from P
    2.2. r ← randomly generate number in range [0, 1].
    2.3. If (r < 0.5)
        2.3.1. Select mutation operations [add conv layer, add skip layer, add pool layer, remove layer of filters], change the value, and position (index value in offspring) randomly.
    2.4. Else
        2.4.1. Choose the crossover point in p1 and p2.
        2.4.2. Apply crossover operation
    2.5. End
3. Return $Q_t$
4. End
```

4.3 Conclusion

After a sequence of evolution, the framework automatically evolves to a suitable CNN architecture. During evolution, a random population is initialized using a predefined encoding and population size. The hyperparameters are manually chosen using the existing state-of-the-art model. Each individual's fitness, which encodes a specific CNN architecture, is assessed throughout evolution using the provided dataset. In following generations, parental individuals are chosen based on their fitness, and the creation of more offspring is assisted through the implementation of genetic operators, such as crossover and mutation. The recently generated populace combines with the preexisting population to form a novel inventory of descendants. The process of evolution continues until the counter surpasses the predetermined maximum generation, at which juncture the counter is incremented by one. we employed adaptive exit conditions that terminate automatically when the convergence rate is very slow or close to zero.

CHAPTER -5

EXPERIMENTAL RESULTS

5.1 Introduction

This section provides a concise overview of the differences between the proposed methodology and the outcomes seen by other researchers in the field. A comparison analysis was performed to evaluate our findings in relation to the current state-of-the-art approaches. The evaluation criteria included classification accuracy, computational resources utilized (measured in terms of GPU days), and architectural considerations. The term "GPU day" refers to the duration of algorithm execution on a single GPU, serving as a metric for quantifying the computational resources utilized by these methods. The outcomes of a comparison between the suggested algorithm and its peer rivals are presented in Table 5.1. The initial column presents a compilation of architectural classifications. The second column has the nomenclature of the architectural structures. The encoding methods are represented in the third column, while the fourth column denotes the evolutionary algorithm employed. The datasets employed are specified in the fifth column, while the classification accuracy is documented in the sixth column. The number of generations is specified in the seventh column, followed by the training epochs in the eighth column. Finally, the parameter count for the pertinent convolutional neural network (CNN) is displayed in the ninth column. Furthermore, the tenth column displays the number of GPU days utilized. All competitors' results in the table are extracted from the related publications; "–" denotes that the results have not been published.

Table 5.1 The classification accuracy comparison on the CIFAR-10 datasets between the proposed algorithm and its contemporary contemporaries [102].

Reference	**Architecture**	**Encoding**	**EA**	**Dataset**	**Accuracy**	**Gen**	**Epoch**	**Parameters**	**GPU**	**Manual Assistance**
[96]	Eden	Fixed Length	PSO	CIFAR10	74.5 %	10	13	1.8 M	12 hour	Partially Required
				MNIST	98.4 %					
[6]	C-PSO-CNN (AlexNet)	Fixed Length	PSO	CIFAR10	89.99 %	40	10	-	-	Partially Required
[6]	C-PSO-CNN (VGG Net-16)	Fixed Length	PSO	CIFAR10	91.02 %	40	10	-	-	Partially Required
[84]	LDPSO	Fixed Length	PSO	CIFAR10	69.37 %	-	10	-	2.37 hour	Partially Required
[7]	Genetic CNN	Fixed Length	GA	CIFAR10	77.06 %	50	-	-	17 days	Partially Required
[97]	EAS	-	RL	CIFAR10	95.77 %	-	300	23.4 M	10 days	Partially Required
[84]	CGP-CNN	variable Length	GP	CIFAR10	94.02 %	50	500	1.68 M	27 days	Not Required
[86]	NAS	-	LSTM	CIFAR10	93.99 %	-	50	2.5 M	22,400 days	Not Required
[85]	E-CNN-MP	Variable Length	GA	MNIST	98.94 %	5	-	-	-	Not Required
[1]	CNN-GA	Variable Length	GA	CIFAR10	77.50 %	5	350	-	-	Not Required
					95.22	20		2.9 M	30 days	
[100]	DCNN	Variable Length	GA	CIFAR10	89.32	10	100	-	12 days	Not Required
				MNIST	99.64	10	100		3 days	

				Fashion_MNIST	94.60	10	100		5 days	
	Proposed Methods	**Variable Length**	**GA**	**CIFAR10**	**79.41 %**	**5**	**40**	**1.2 M**	**2.47 hour**	**Not Required**
					87.02 %	**10**	**40**	**1.6 M**	**6.38 hour**	
				MNIST	**99.39 %**	**10**	**10**	**2.7 M**	**3.12 hour**	
				Fashion_MNIST	**93.07 %**	**10**	**10**	**1.4 M**	**3.27 hour**	

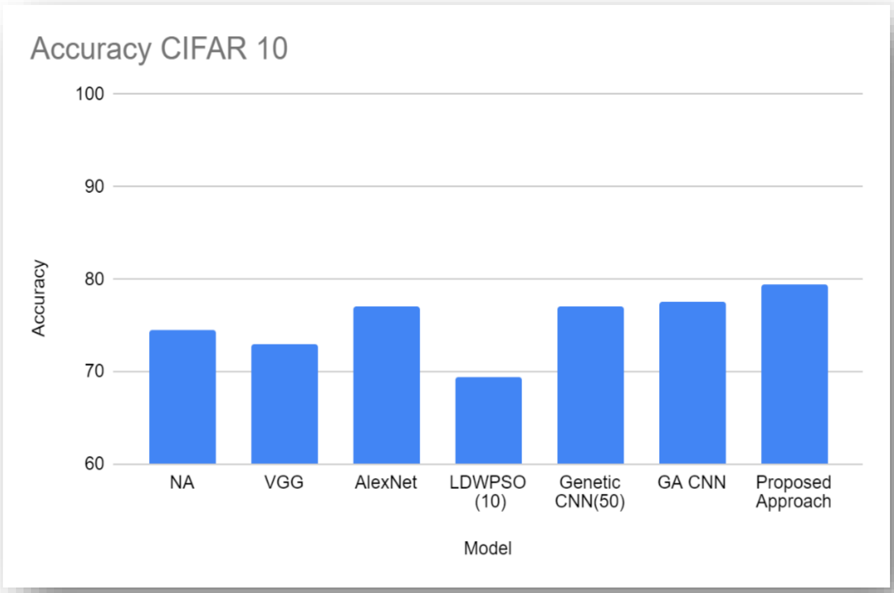


Figure 5.1 Comparison between different evolutionary CNN algorithms using CIFAR-10 datasets.

5.2 Discussion

Table 5.1 displays the results of a comparison between the proposed algorithm and its peer competitors. In this table the peer competitors into two categories. In the first group, we compared architectures requiring manual aid in design selection or parameter adjustment. In this category, the computation cost is less because half of the work is performed by professionals. In the second group, we compared architectures that evolved without human involvement. Our methodology exhibits a significant improvement in classification accuracy on the CIFAR10 dataset within the primary category of peer competitors. Specifically, the Eden architecture exhibits a 16.8% improvement, while the LDPSO design showcases a more substantial increase of 25.4%. In the second category, our technique improves classification accuracy on the CIFAR10 dataset by 2.46% for CNN-GA architecture and 12.8% for Genetic-CNN architecture as shown in Figure 5.1 for 5 generation and in Figure 5.4 for 10 generation respectively. It also exhibits a 1.06% and 0.45% improvement on the MNIST dataset over the Eden and E-CNN-MP architectures, respectively. We further tested the efficacy of our approach using the Fashion_MNIST dataset, which demonstrates a 3.37% improvement over the capsuleNet [99] architecture. The classification accuracy is slightly less for EAS, CGP-CNN, NAS, CNN-GA, and DCNN architecture. However, in addition to classification, we also compare the effectiveness based on the number of parameters, epoch, and computation required in GPU. DCNN requires 100 epochs and 12 GPU days to train CIFAR-10 datasets, CNN-GA requires 30 GPU days and up to 350 epochs, CGP-CNN is trained in 500 epochs and requires 27 GPU days, EAS requires 10 GPU days with 300 epochs, and NAS requires 50 epoch and 22,400 GPU days. Our algorithm was trained in 10 generations, 10 epochs, and 6.38 hours using the Nvidia A100 GPU configuration on the CIFAR 10 dataset, demonstrating significant improvement and a faster convergence rate. Our techniques provide competitive performance in terms of precision and the number of parameters while requiring less calculation time.

To assess the efficacy of the proposed methodology in revealing the structure of the convolutional neural network (CNN), we have presented the evolution of outcomes in Tables 5.2 and Table 5.3, corresponding to the MNIST and Fashion_MNIST datasets, respectively. Figure 5.2 illustrates the evolutionary history of the proposed approach in its pursuit of identifying the optimal convolutional neural network (CNN) architecture for the MNIST dataset. The figure presents two key metrics: (a) average accuracy and (b) top accuracy. Additionally, Figure 5.3 illustrates the evolutionary trajectory of the proposed method in its pursuit of identifying the optimal architecture of a Convolutional Neural Network (CNN) on the CIFAR10 dataset. The effectiveness of the proposed algorithm is demonstrated by comparing the convergence of various benchmark evolutionary trajectories, as depicted in Figure 5.5. We randomly chose the N population and initialized the value using the recommended encoding approach. In this experiment, the population size is N=5, the number of epochs is 40, and the number of generations is 10. The architectures are trained using the backpropagation technique for forty epochs. In the training and validation sets, we utilized an 80/20 ratio. After training, the validation accuracy applied in the fitness function for decision-making of the proposed method is calculated. We eliminated weaker populations for the next generation using mode accuracy. The fitness function eliminates 50 % of the population in each generation, and the best 50 % are employed as parents. The selected population is repopulated using the genetic operator's mutation and crossover with probabilities of 0.2 for mutation and 0.80 for crossover. The existing fittest population is merged with the new population. Thus, using this method, the same number of chromosomes is available in each generation. The effectiveness of the suggested algorithm has been assessed through the utilization of statistical measures such as the minimum, mean, maximum, mode, and standard deviation, and standard error of the mean (SEM). The standard error of the mean (SEM) measures how much discrepancy is likely in a sample's mean compared with the population mean. For each iteration, we have selected the most improved CNN architecture, which is represented in the final column of Table 5.2 and Table 5.3. The maximum accuracy indicates the best accuracy obtained by any CNN architecture at that generation. The standard deviation demonstrates the genetic algorithm's efficacy in terms of a quicker convergence rate. Initial standard deviation values are largely due to the random initialization of the population. However, its value decreases over successive generations, and the top accuracy rises, bringing the outcome close to the global optimum. If the standard deviation continues to decline, the subsequent few generations will see greater convergence. With this strategy, we reached a standard near the benchmark accuracy in 10 epochs and 10 generations, demonstrating a faster convergence rate with equivalent precision and less computational power. This technique yielded 99.39% top accuracy on the MNIST datasets and 93.07% on the Fashion_MNIST dataset, equivalent to the benchmark accuracy without user intervention and requiring less GPU days. The evolutionary progression of the algorithm under consideration in uncovering the optimal convolutional neural network (CNN) structure on both the Fashion MNIST dataset and MNIST dataset is depicted in Figure 5.6 and Figure 5.8 correspondingly. Figure 5.7 illustrates the comparison of various state-of-the-art architectures for the MNIST dataset.

Table 5.2 Evolution of CNN model using MNIST dataset with population size=5, epoch 40, and generation=10

Generation	**Min%**	**Avg %**	**Max%**	**Med%**	**Std-D**	**SME**	**Best CNN model**
Gen 1	87.15	95.84	99.15	97.46	4.39	1.96	256-512-max-max-512-256
Gen 2	97.46	98.34	99.15	98.01	0.68	0.30	256-512-max-max-512-256
Gen 3	97.49	98.69	99.15	99.15	0.66	0.29	256-512-max-max-512-256
Gen 4	98.57	99.01	99.23	99.15	0.24	0.10	256-512-max-512-256-max-512-256
Gen 5	98.62	99.04	99.23	99.15	0.21	0.09	256-512-max-512-256-max-512-256
Gen 6	99.15	99.21	99.33	99.23	0.06	0.04	256-512-max-max-256-512-max-512-256-max-512-256

Gen 7	99.23	99.27	99.33	99.23	0.05	0.02	256-512-max-max-256-512-max-512-256-max-512-256
Gen 8	99.04	99.26	99.36	99.33	0.11	0.05	256-512-max-max-512-256-max-512-256
Gen 9	99.10	99.29	99.36	99.36	0.04	0.10	256-512-max-max-512-256-max-512-256
Gen 10	99.29	99.34	99.39	99.36	0.02	0.01	256-512-512-256-max-512-256-max-512-256

Table 5.3 Evolution of CNN model using Fashion_MNIST dataset with population size=5, epoch 10, and generation=10

Generation	**Min %**	**Avg %**	**Max%**	**Med%**	**Std-D**	**SME**	**Best CNN model**
Gen 1	88.20	89.25	91.01	88.69	0.99	0.44	128-256-512-256-mean-mean
Gen 2	88.69	89.44	91.01	89.12	0.86	0.38	128-256-512-256-mean-mean
Gen 3	89.12	89.77	91.01	89.64	0.67	0.29	128-256-512-256-mean-mean
Gen 4	89.59	90.70	91.43	91.01	0.70	0.31	128-256-512-256-mean-mean-mean
Gen 5	91.01	91.35	91.60	91.40	0.19	0.08	128-256-512-256-512-256-mean-mean
Gen 6	91.01	91.35	91.60	91.40	0.19	0.08	128-256-512-256-512-256-mean-mean
Gen 7	91.40	91.86	93.07	91.60	0.24	0.11	128-256-128-256-mean-mean-mean
Gen 8	91.41	91.94	93.07	91.79	0.58	0.26	128-256-128-256-mean-mean-mean

Gen 9	91.60	91.98	93.07	91.79	0.55	0.24	128-256-128-256-mean-mean-mean
Gen 10	91.64	92.11	93.07	91.82	0.51	0.23	128-256-128-256-mean-mean-mean

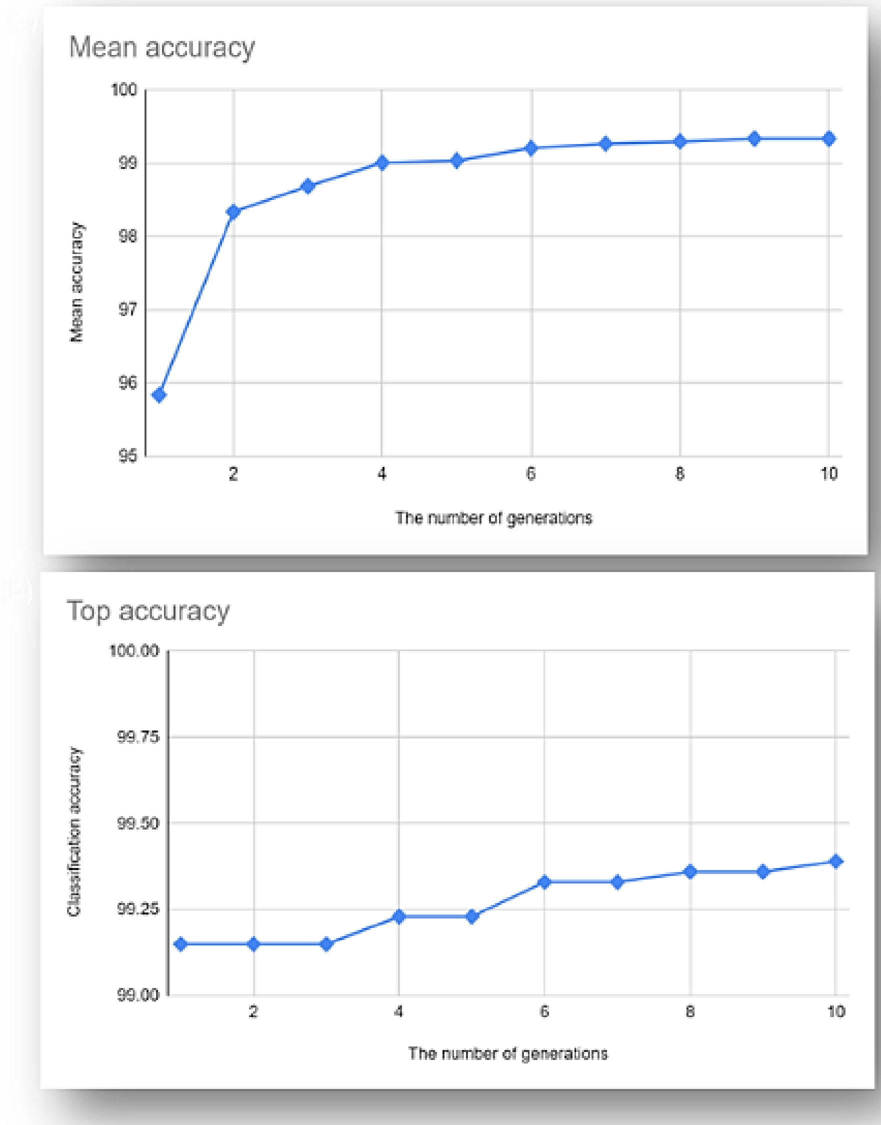


Figure 5.2 The proposed algorithm's evolutionary trajectory in discovering the best CNN architecture on the MNIST dataset; (a) Avg accuracy (b) Top accuracy.

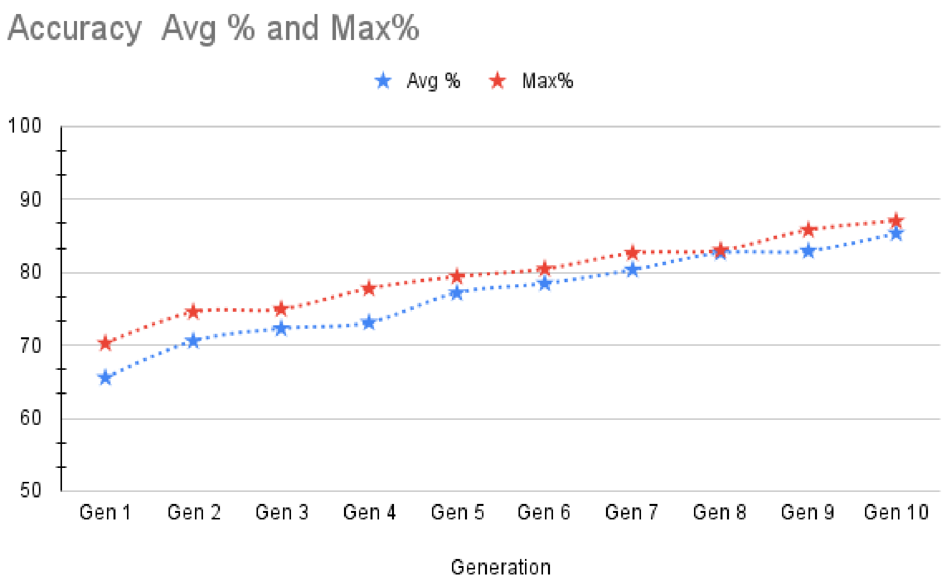


Figure 5.3 The trajectory of the proposed algorithm as it discovers the optimal CNN architecture on the CIFAR10 dataset.

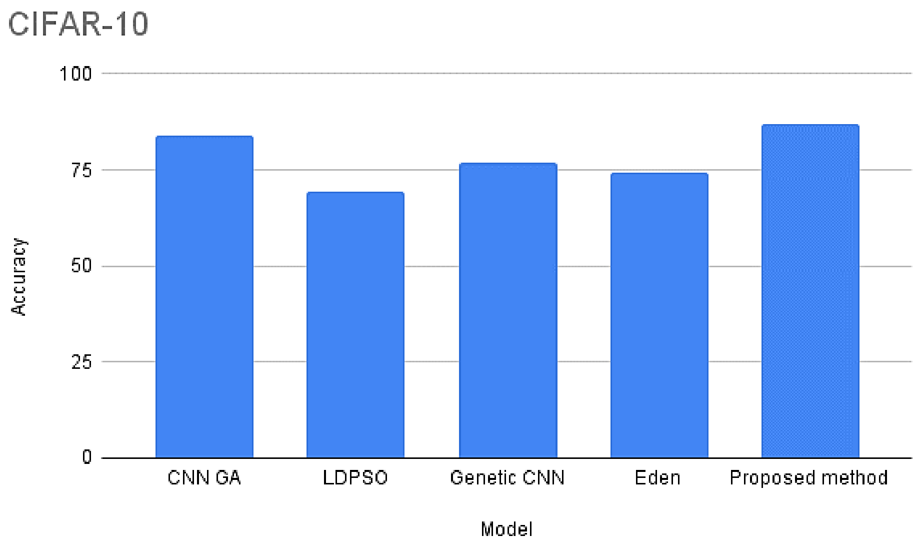


Figure 5.4 Comparison between different evolutionary CNN algorithms using CIFAR-10 datasets

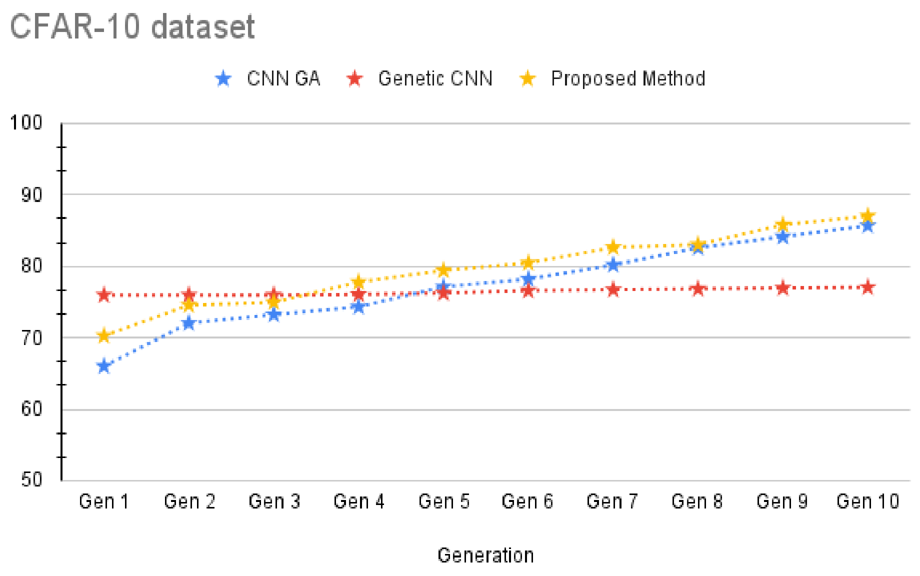


Figure 5.5 Comparing the evolutionary trajectory of the proposed algorithm with CNN-GA and Genetic CNN to determine the optimal CNN architecture for the CIFAR10 dataset.

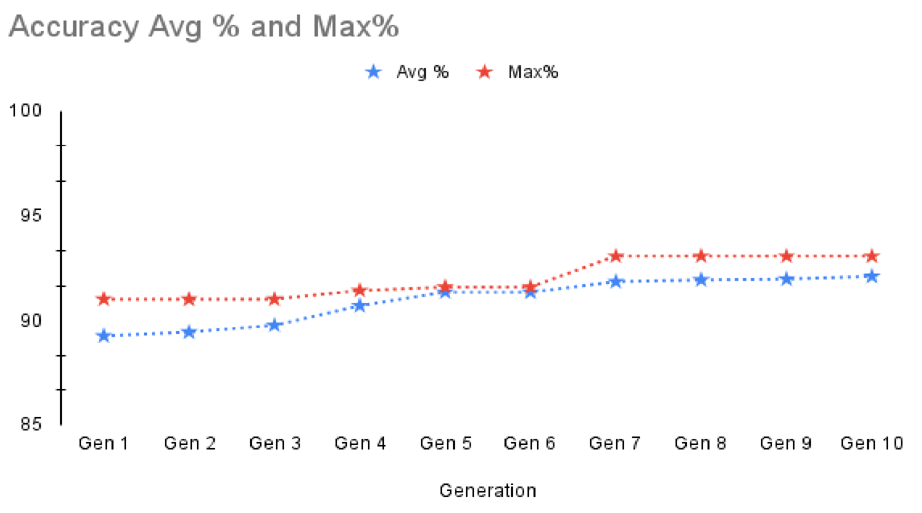


Figure 5.6 The evolutionary trajectory of the proposed algorithm in discovering the best architecture of CNN on the Fashion MNIST dataset

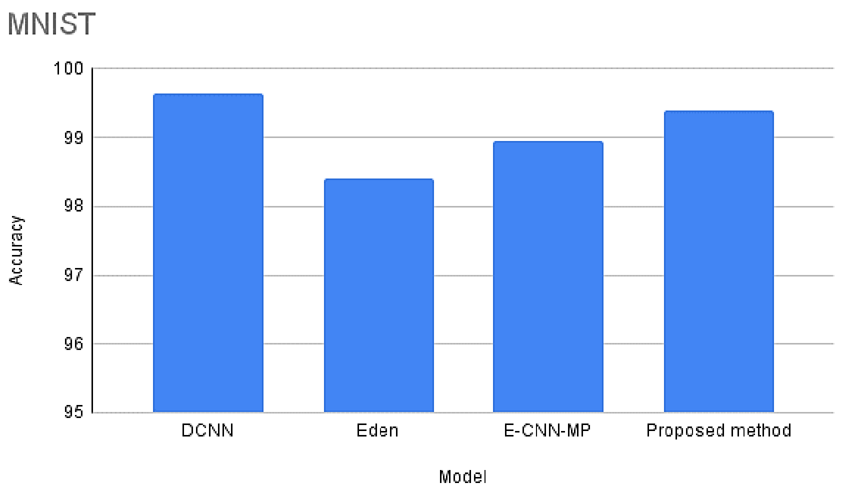


Figure 5.7 Comparison between different evolutionary CNN algorithms using MNIST datasets

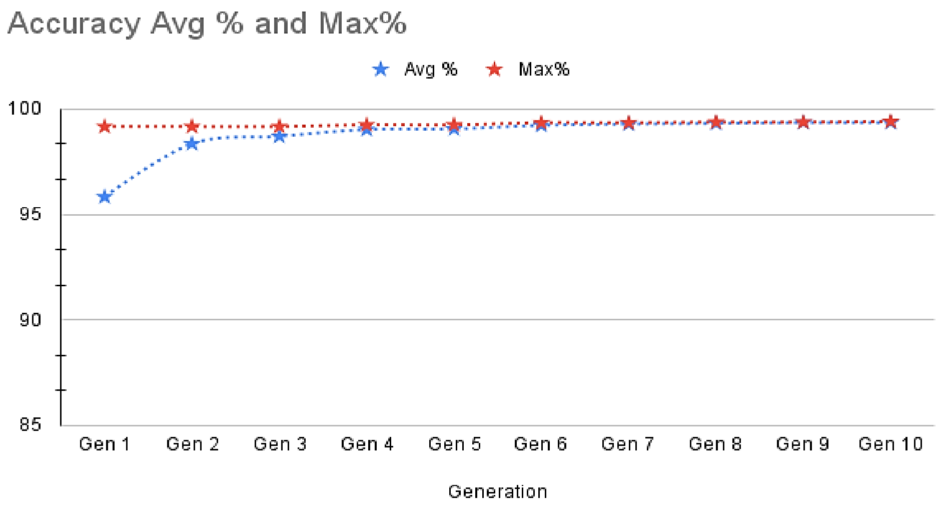


Figure 5.8 The trajectory of the proposed algorithm as it discovers the optimal CNN architecture on the MNIST dataset.

The proposed algorithm demonstrated an increase in convergence rate using the defined techniques. We have displayed the evolution in Table 5.2 and Table 5.3 to help understand the efficacy of the suggested approach for discovering CNN designs. We used the MNIST and Fashion_MNIST datasets with a population size of 5, where each architecture is trained for 40 and 10 epochs, respectively. The evolution of each generation is expressed using standard deviation and top accuracy. It shows the effectiveness of the proposed algorithm with its faster convergence rate in the initial iteration reaching toward global optima without being stuck in the local one. Additionally, it offers the diversity of the algorithm that works suitably in different datasets**.**

5.3 Conclusion

This study aims to develop a genetic algorithm (GA)-based method for automatically designing convolutional neural network (CNN) architectures. This method seeks to select the most suitable CNN architecture for image classification tasks, specifically targeting users who have limited expertise in changing CNN structures. The objective was achieved by the presentation of a new encoding method for the genetic algorithm (GA) that allows for the encoding of various depths for convolutional neural networks (CNNs). The proposed technique is assessed and contrasted with 11 contemporary peer competitors, comprising of four partial tuning and seven automatic algorithms that determine the architectures of CNNs. The experimental results obtained from the MNIST, Fashion_MNIST, and CIFAR10 datasets provide evidence that the suggested methodology possesses the potential to autonomously generate deep convolutional neural network (DCNN) structures that are equivalent to, and in some cases, even surpass state-of-the-art models.

CHAPTER -6

CONCLUSION AND FUTURE SCOPE

The utilization of a genetic algorithm for the automatic selection of convolutional neural network (CNN) architecture has demonstrated efficacy in optimizing the performance of such networks. The genetic algorithm (GA) is a search technique that draws inspiration from natural selection and genetics. It is employed to iteratively refine a population of potential solutions, aiming to converge towards an ideal answer. To use a genetic algorithm for automatic CNN architecture selection define the objective of the CNN architecture selection, such as maximizing classification accuracy on a specific dataset. In the context of CNN architecture, a chromosome represents a candidate architecture. We need to define a suitable chromosome representation, which can be a binary string or a list of integers representing different architectural choices (e.g., number of layers, filter sizes, pooling operations). Generate an initial population of random candidate architectures (chromosomes). The size of the population is contingent upon the intricacy of the search space and the availability of computer resources. Assess the level of suitability of each potential architecture within the population in terms of fitness. Train and evaluate each architecture on a validation set using a predefined fitness function (e.g., classification accuracy). The fitness function measures how well each architecture performs on the given objective. The process of selecting individuals from the population is conducted based on their fitness scores. Greater levels of physical fitness are indicative of superior performance. Various selection methods commonly employed in evolutionary algorithms include tournament selection, roulette wheel selection, and rank-based selection. The application of genetic operators, such as crossover and mutation, is employed to generate novel candidate designs derived from the chosen people. The process of crossover involves the amalgamation of genetic material from two parent architectures in order to generate offspring, whereas mutation brings minor and random alterations to the design. The process involves doing the evaluation, selection, and genetic operators repeatedly for a predetermined number of generations or until a termination requirement is satisfied. This termination criterion could be achieving a desirable fitness level or depleting computational resources. Upon the algorithm reaching the termination criterion, the optimal Convolutional Neural Network (CNN) architecture is determined by selecting the architecture that exhibits the highest performance from the final population.

6.1 Summary of the Thesis

In this thesis, we presented a methodology to design an effective CNN model using evolutionary algorithms. This framework generates a CNN architecture automatically using the datasets given. We validated our proposed methods using three benchmark datasets MNIST, Fashion_MNIST, and CFAR-10. These datasets were utilized for training and evolving our deep learning models. We compared our methods with the existing state-of-the-art architectures using parameters accuracy, computation cost, and the number of parameters obtained. The main research objectives mentioned in Chapter 1, section 1.3 have been addressed in this thesis in the following order:

First, we propose two novel encoding representations: Each encoding representation outlines the process of population initialization. The length of the individual, which signifies the depth of the related Convolutional Neural Network (CNN), is initialized in a random manner. The initial step involves the creation of a linked list of L nodes, with each node being appropriately specified. If a number generated at random is found to be less than 0.5, the corresponding node is designated as a skip layer. Alternatively, it can be interpreted as a pooling layer, with the specific pooling type being selected by an additional random variable. In skip connection layers, the feature map numbers for the nodes are assigned in a random manner. An illustrative demonstration of the suggested encoding approach for a Convolutional Neural Network (CNN) is presented, showcasing the depiction of skip layers and pooling layers through designated codes. The whole convolutional neural network (CNN) architecture is represented by a sequential concatenation of codes corresponding to the individual layers. For instance, a CNN with a depth of 8 can be denoted as "32-64-0.2-64-256-0.8-512-256".

The second scheme employs a variable-length encoding approach that represents both the depth and width of the architecture. It consists of four building blocks: the genesis block, transit block, agile block, and fully connected blocks. The genesis block handles the input image size, while the transit block reduces feature maps and dimensions using 1x1 convolution and pooling operations. The value of the pooling operation determines whether max pooling or mean pooling is used. The agile block incorporates dense connections and skip connections to reduce parameters and increase complexity. The components encompassed under this framework consist of operational procedures, the dimensions of the filter, the quantity of filters employed, the level of depth, and the interconnections established between convolutional layers. Ultimately, the fully linked blocks serve to flatten the layer and subsequently transform it into the output layer, which is characterized by the number of classes. The proposed encoding scheme offers several advantages. It enables the representation of architectures with a combination of two different layers, simplifying the representation and facilitating the increase in architecture depth. The scheme supports evolutionary operations like mutation and crossover efficiently due to its fewer parameters. It also allows for increasing complexity within the blocks, randomly generating filter sizes and depths in the agile block. The scheme incorporates a hybrid encoding scheme, utilizing binary and decimal representations. It provides flexibility for exploring depth and width, leading to faster optimization.

Third, we demonstrate a novel framework to evolve the CNN architecture automatically using GA. In this algorithm, we pass input datasets, and after a sequence of evolution, the framework automatically evolves to a suitable CNN architecture. A random population is initialized using a predetermined encoding and population size throughout evolution. The basic components of a CNN are convolutional layers, pooling layers, and sometimes fully linked layers. The CNN's performance heavily depends on its parameters, which depend on the connection depth and width**.** The fully connected layer is discarded in this encoding as many parameters make it computationally inefficient. Initially, the number of population and the depth of each population is selected randomly. In the selected population, the first layer is fixed as a convolutional layer; then, convolutional and pooling layers are determined randomly with equal probability. It evaluates the fitness of all input populations using a given dataset. An individual's CNN is initially decoded using a predetermined set of hyperparameter parameters. CNN decoding is trained with training data, and accuracy is used to determine fitness. Because the training of CNN is a time-taking task, we used half of the dataset for initial training to make it efficient. After training the population, half of the population is eliminated based on fitness score. The best population is chosen for reproduction in the following offspring generation. Specifically, two parents are selected based on which of two randomly selected individuals is more suitable. We build a new set of populations with equal probability by utilising mutation and crossover processes. In a crossover operation, each parent is arbitrarily divided into two pieces, and the two pieces from each parent are exchanged to generate two offspring. After a sequence of evolution, the framework automatically evolves to a suitable CNN architecture. During evolution, a random population is initialized using a predefined encoding and population size. The hyperparameters are manually chosen using the existing state-of-the-art model. Each individual's fitness, which encodes a specific CNN architecture, is assessed throughout evolution using the provided dataset.

In subsequent generations, parental individuals are selected based on their fitness, and the production of additional offspring is facilitated through the implementation of genetic operators, including crossover and mutation. The recently generated population is merged with the preexisting population to form a novel roster of progeny. The process of evolution continues until the counter surpasses the maximum generation threshold, at which juncture the counter is incremented by one. The primary aim of this research is to devise a methodology for the automated generation of Convolutional Neural Networks (CNNs) through the utilization of Genetic Algorithms (GA). This technique seeks to determine the most effective CNN architecture for image classification tasks, specifically targeting users who have limited expertise in changing CNN structures. The objective was achieved by the proposal of a novel encoding strategy for the genetic algorithm (GA) to encode arbitrary depths of convolutional neural networks (CNNs). The proposed methodology is assessed and contrasted with 11 contemporary peer rivals, comprising of four partial tuning and seven automatic techniques used for establishing the architectures of CNNs. The empirical findings obtained from conducting experiments on the MNIST, Fashion_MNIST, and CIFAR10 datasets demonstrate that the proposed methodology has the capability to autonomously create deep convolutional neural network (DCNN) architectures that are on par with, or perhaps beyond, the most advanced models currently available.

6.2 Future Work

The proposed framework for automatic CNN architecture evolution using genetic algorithm for image classification provides better classification accuracy than the baseline model due to the effectiveness of encoding representation and proposed genetic operators in obtaining better accuracy in benchmark datasets MNIST, CIFAR10, and Fashion_MNIST. However, the research presented here have wider scope with several extensions addressing variety of challenges that require future attention, as is the case with many other academic articles in the same field. In the part that follows, we go through some of these issues and suggest upcoming directions that, in our opinion, will have a significant influence.

The proposed work focus on linear variable length encoding scheme that make its application easier to adopt and implement using different genetic operators.

But it cannot handle the skip connection in deeper architecture. Therefore, In deep neural networks, as the gradient flows backward during the training process, it can get progressively smaller, leading to the vanishing gradient problem. When gradients become too small, the network struggles to update the weights of earlier layers effectively, making training difficult. Skip connections facilitate the integration of features originating from various depths within the network. This enables the network to integrate low-level and high-level elements, hence enabling the acquisition of intricate patterns and capturing both detailed and overarching information. Additionally in existing encoding we considered only convolutional layer and pooling layer but fully connected layer is skiped. In future work fully connected layer also be part of improved architecture that improve the accuracy,

The proposed work performance is mostly on lightweight CNN architectures, that often perform well on small datasets due to their simplicity and efficiency. These architectures are designed to have fewer parameters and lower computational requirements, making them suitable for small datasets and resource-constrained environments. However, when applied to large datasets, they might not be as effective. Lightweight CNN architectures usually have a smaller number of layers and parameters compared to larger, more complex architectures. While this simplicity allows them to be trained on small datasets, it also limits their capacity to capture and represent complex patterns in large datasets. Large datasets often contain diverse and intricate patterns that require more complex models to be effectively learned. To perform well on large datasets, CNN architectures with higher capacity, more layers, and more parameters are often preferred. Such architectures have the potential to learn more complex representations, capture intricate patterns, and generalize better to the diversity present in large datasets.

Further research could investigate the technique's efficacy in various evolutionary algorithms that can accelerate the CNN fitness measurement. The proposed work is based on GA. It is known for its ability to explore a broader search space, making it more effective at global optimization, especially for complex and multimodal problems. Whereas other algorithms such as PSO [103-104], and DE [105] on the other hand, tends to be more exploitative, meaning it is better at fine-tuning solutions once a good region of the search space is found. Additionally, variations of both PSO and GA, and hybrid approaches combining elements of both algorithms have been proposed to leverage their strengths while mitigating their weaknesses.

REFERENCES

[1] Sun, Y., Xue, B., Zhang, M., Yen, G. G., & Lv, J. (2020). Automatically designing CNN architectures using the genetic algorithm for image classification. *IEEE transactions on cybernetics,* 50(9), 3840-3854.

[2] Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images.

[3] Deng, L. (2012). The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine,* 29(6), 141-142.

[4] Galván, E., & Mooney, P. (2021). Neuroevolution in deep neural networks: Current trends and future challenges. *IEEE Transactions on Artificial Intelligence,* 2(6), 476-493.

[5] Esfahanian, P., & Akhavan, M. (2019). Gacnn: Training deep convolutional neural networks with genetic algorithm. *arXiv preprint arXiv:*1909.13354.

[6] Suganuma, M., Kobayashi, M., Shirakawa, S., & Nagao, T. (2020). Evolution of deep convolutional neural networks using cartesian genetic programming. *Evolutionary computation,* 28(1), 141-163.

[7] Xie, L., & Yuille, A. (2017). Genetic cnn. In *Proceedings of the IEEE international conference on computer vision* (pp. 1379-1388).

[8] Bengio, Y., & LeCun, Y. (2007). Scaling learning algorithms towards AI. *Large-scale kernel machines,* 34(5), 1-41.

[9] Girshick, R. (2015). Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision* (pp. 1440-1448).

[10] Barik, D., & Mondal, M. (2010, June). Object identification for computer vision using image segmentation. In *2010 2nd International conference on education technology and computer* (Vol. 2, pp. V2-170). IEEE.

[11] Muthukrishnan, R., & Radha, M. (2011). Edge detection techniques for image segmentation. *International Journal of Computer Science & Information Technology*, 3(6), 259.

[12] Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13* (pp. 818-833). Springer International Publishing.

[13] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... & Hassabis, D. (2016). *Mastering the game of Go with deep neural networks and tree search. nature,* 529(7587), 484-489.

[14] Sinha, T., Haidar, A., & Verma, B. (2018, July). Particle swarm optimization based approach for finding optimal values of convolutional neural network parameters. In *2018 IEEE congress on evolutionary computation (CEC)* (pp. 1-6). IEEE.

[15] Lorenzo, P. R., Nalepa, J., Kawulok, M., Ramos, L. S., & Pastor, J. R. (2017, July). Particle swarm optimization for hyper-parameter selection in deep neural networks. In *Proceedings of the genetic and evolutionary computation conference* (pp. 481-488).

[16] Yamasaki, T., Honma, T., & Aizawa, K. (2017, April). Efficient optimization of convolutional neural networks using particle swarm optimization. In *2017 IEEE third international conference on multimedia big data (BigMM)* (pp. 70-73). IEEE.

[17] Soon, F. C., Khaw, H. Y., Chuah, J. H., & Kanesan, J. (2018). Hyper-parameters optimisation of deep CNN architecture for vehicle logo recognition. *IET Intelligent Transport Systems*, 12(8), 939-946.

[18] Wang, B., Sun, Y., Xue, B., & Zhang, M. (2018, July). Evolving deep convolutional neural networks by variable-length particle swarm optimization

for image classification. In *2018 IEEE Congress on Evolutionary Computation (CEC)* (pp. 1-8). IEEE.

[19] Talathi, S. S. (2015, September). Hyper-parameter optimization of deep convolutional networks for object recognition. In *2015 IEEE international conference on image processing (ICIP)* (pp. 3982-3986). IEEE.

[20] Fan, E. (2000). Extended tanh-function method and its applications to nonlinear equations. *Physics Letters A*, 277(4-5), 212-218.

[21] Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)* (pp. 807-814).

[22] Han, J., & Moraga, C. (1995, June). The influence of the sigmoid function parameters on the speed of backpropagation learning. In *International workshop on artificial neural networks* (pp. 195-201). Berlin, Heidelberg: Springer Berlin Heidelberg.

[23] Zunino, R., & Gastaldo, P. (2002, May). Analog implementation of the softmax function. In *2002 IEEE international symposium on circuits and systems. Proceedings (Cat. No. 02CH37353)* (Vol. 2, pp. II-II). IEEE.

[24] Xiao, S., Li, T., & Wang, J. (2020). Optimization methods of video images processing for mobile object recognition. *Multimedia Tools and Applications*, 79, 17245-17255.

[25] Joshi, D., & Singh, T. P. (2020). A survey of fracture detection techniques in bone X-ray images. *Artificial Intelligence Review*, 53(6), 4475-4517.

[26] Bacanin, N., Bezdan, T., Tuba, E., Strumberger, I., & Tuba, M. (2020). Optimizing convolutional neural network hyperparameters by enhanced swarm intelligence metaheuristics. *Algorithms*, 13(3), 67.)

[27] Mendoza, H., Klein, A., Feurer, M., Springenberg, J. T., & Hutter, F. (2016, December). Towards automatically-tuned neural networks. In *Workshop on automatic machine learning* (pp. 58-65). PMLR.

[28] Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 8697-8710).

[29] Liu, H., Simonyan, K., Vinyals, O., Fernando, C., & Kavukcuoglu, K. (2017). Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436.*

[30] Ren, J., Li, Z., Yang, J., Xu, N., Yang, T., & Foran, D. J. (2019). Eigen: Ecologically-inspired genetic approach for neural network structure searching from scratch. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 9059-9068).

[31] Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., ... & Kurakin, A. (2017, July). Large-scale evolution of image classifiers. In *International conference on machine learning* (pp. 2902-2911). PMLR.

[32] Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2), 99-127.

[33] Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L. J., ... & Murphy, K. (2018). Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)* (pp. 19-34).

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

[35] Mirjalili, S., & Mirjalili, S. (2019). Genetic algorithm. *Evolutionary Algorithms and Neural Networks: Theory and Applications*, 43-55.

[36] Kennedy, J., & Eberhart, R. (1995, November). Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks* (Vol. 4, pp. 1942-1948). IEEE.

[37] Shirani Faradonbeh, R., Monjezi, M., & Jahed Armaghani, D. (2016). Genetic programing and non-linear multiple regression techniques to predict backbreak in blasting operation. *Engineering with computers*, 32, 123-133.

[38] Wang, B., Sun, Y., Xue, B., & Zhang, M. (2018). A hybrid differential evolution approach to designing deep convolutional neural networks for image classification. In *AI 2018: Advances in Artificial Intelligence: 31st Australasian Joint Conference, Wellington, New Zealand, December 11-14, 2018, Proceedings 31* (pp. 237-250). Springer International Publishing.

[39] Elsken, T., Metzen, J. H., & Hutter, F. (2017). Simple and efficient architecture search for convolutional neural networks. *arXiv preprint arXiv:1711.04528.*

[40] Liu, H., Simonyan, K., & Yang, Y. (2018). Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.

[41] Pham, H., Guan, M., Zoph, B., Le, Q., & Dean, J. (2018, July). Efficient neural architecture search via parameters sharing. In *International conference on machine learning* (pp. 4095-4104). PMLR.

[42] Khan, A., Sohail, A., Zahoora, U., & Qureshi, A. S. (2020). A survey of the recent architectures of deep convolutional neural networks. *Artificial intelligence review*, 53, 5455-5516.

[43] Fukushima, K. (1988). Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural networks*, 1(2), 119-130.

[44] LeCun, Y., Jackel, L. D., Bottou, L., Cortes, C., Denker, J. S., Drucker, H., ... & Vapnik, V. (1995). Learning algorithms for classification: A comparison on handwritten digit recognition. *Neural networks: the statistical mechanics perspective,* 261(276), 2.

[45] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems,* 25.

[46] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv*:1409.1556.

[47] Wu, S., Zhong, S., & Liu, Y. (2018). Deep residual learning for image steganalysis. *Multimedia tools and applications*, 77, 10437-10453.

[48] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1-9).

[49] Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1251-1258).

[50] Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems,* 6(02), 107-116.

[51] Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4700-4708).

[52] Wu, H., & Gu, X. (2015). Towards dropout training for convolutional neural networks. *Neural Networks,* 71, 1-10.

[53] Keskar, N. S., & Socher, R. (2017). Improving generalization performance by switching from adam to sgd. *arXiv preprint arXiv:1712.07628*.

[54] Bock, S., Goppold, J., & Weiß, M. (2018). An improvement of the convergence proof of the ADAM-Optimizer. *arXiv preprint arXiv:1804.10587*.

[55] Lydia, A., & Francis, S. (2019). Adagrad—an optimizer for stochastic gradient descent. *Int. J. Inf. Comput. Sci*, 6(5), 566-568.

[56] Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701.*

[57] Kurbiel, T., & Khaleghian, S. (2017). Training of deep neural networks based on distance measures using RMSProp. *arXiv preprint arXiv:1708.01911.*

[58] Zhang, Z., & Sabuncu, M. (2018). Generalized cross entropy loss for training deep neural networks with noisy labels. *Advances in neural information processing systems*, 31.

[59] Ma Cao, J., Su, Z., Yu, L., Chang, D., Li, X., & Ma, Z. (2018, November). Softmax cross entropy loss with unbiased decision boundary for image classification. In *2018 Chinese automation congress (CAC)* (pp. 2028-2032). IEEE.

[60] Das, K., Jiang, J., & Rao, J. N. K. (2004). Mean squared error of empirical predictor.

[61] Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009, June). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition* (pp. 248-255). Ieee.

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

[63] Chunna, L., Hai, F., & Chunlin, G. (2020). Development of an efficient global optimization method based on adaptive infilling for structure optimization. *Structural and Multidisciplinary Optimization*, 62, 3383-3412.

[64] Kitjacharoenchai, P., Ventresca, M., Moshref-Javadi, M., Lee, S., Tanchoco, J. M., & Brunese, P. A. (2019). Multiple traveling salesman problem with drones: Mathematical model and heuristic approach. *Computers & Industrial Engineering*, 129, 14-30.

[65] Lin, M., Chen, Q., & Yan, S. (2013). Network in network. *arXiv preprint arXiv:1312.4400.*

[66] Lucas, S. (2021). The origins of the halting problem. *Journal of Logical and Algebraic Methods in Programming*, 121, 100687.

[67] Lee, K. S., & Geem, Z. W. (2005). A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice. *Computer methods in applied mechanics and engineering*, 194(36-38), 3902-3933.

[68] Voß, S., Martello, S., Osman, I. H., & Roucairol, C. (Eds.). (2012). Meta-heuristics: Advances and trends in local search paradigms for optimization.

[69] Beyer, H. G., & Schwefel, H. P. (2002). Evolution strategies–a comprehensive introduction. *Natural computing*, 1, 3-52.

[70] Hansen, N., Arnold, D. V., & Auger, A. (2015). Evolution strategies. *Springer handbook of computational intelligence*, 871-898.

[71] Koza, J. R. G. P. (1992). On the programming of computers by means of natural selection. *Genetic programming*.

[72] Michalewicz, Z., & Schoenauer, M. (1996). Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary computation*, 4(1), 1-32.

[73] Yu, X., & Gen, M. (2010). Introduction to evolutionary algorithms. *Springer Science & Business Media*.

[74] Dorigo, M., Birattari, M., Di Caro, G. A., Doursat, R., Engelbrecht, A. P., Floreano, D., ... & Sayama, H. (Eds.). (2010). *Swarm Intelligence: 7th International Conference, ANTS 2010, Brussels, Belgium, September 8-10, 2010 Proceedings* (Vol. 6234). Springer.

[75] Hu, X., Eberhart, R. C., & Shi, Y. (2003, April). Swarm intelligence for permutation optimization: a case study of n-queens problem. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No. 03EX706)* (pp. 243-246). IEEE.).

[76] Karaboga, D. (2005). An idea based on honey bee swarm for numerical optimization (Vol. 200, pp. 1-10). *Technical report-tr06, Erciyes university, engineering faculty, computer engineering department*.

[77] Houck, C. R., Joines, J., & Kay, M. G. (1995). A genetic algorithm for function optimization: a Matlab implementation. *Ncsu-ie tr*, 95(09), 1-10.

[78] Le,Qin, A. K., Huang, V. L., & Suganthan, P. N. (2008). Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE transactions on Evolutionary Computation*, 13(2), 398-417.

[79] Yao, X., Liu, Y., & Lin, G. (1999). Evolutionary programming made faster. *IEEE Transactions on Evolutionary computation*, 3(2), 82-102.

[80] Sun, Y., Xue, B., Zhang, M., & Yen, G. G. (2019). Completely automated CNN architecture design based on blocks. *IEEE transactions on neural networks and learning systems*, 31(4), 1242-1254.

[81] Beyer, H. G., & Schwefel, H. P. (2002). Evolution strategies–a comprehensive introduction. *Natural computing*, 1, 3-52.

[82] Sun, Y., Xue, B., Zhang, M., & Yen, G. G. (2019). Evolving deep convolutional neural networks for image classification. *IEEE Transactions on Evolutionary Computation*, 24(2), 394-407.

[83] Wang, Y., Zhang, H., & Zhang, G. (2019). cPSO-CNN: An efficient PSO-based algorithm for fine-tuning hyper-parameters of convolutional neural networks. *Swarm and Evolutionary Computation*, 49, 114-123.

[84] Serizawa, T., & Fujita, H. (2020). Optimization of convolutional neural network using the linearly decreasing weight particle swarm optimization. *arXiv preprint arXiv:2001.05670*.

[85] Loussaief, S., & Abdelkrim, A. (2018). Convolutional neural network hyper-parameters optimization based on genetic algorithms. *International Journal of Advanced Computer Science and Applications*, 9(10).

[86] Zoph, B., & Le, Q. V. (2016). Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578.*

[87] Zhong, Z., & Yan, J. (2018). Liu, Cheng-Lin. Practical network blocks design with q-learning. In *AAAI Conference on Artificial Intelligence*.

[88] Ari Hassanzadeh, T., Essam, D., & Sarker, R. (2020, March). EvoU-Net: An evolutionary deep fully convolutional neural network for medical image segmentation. In *Proceedings of the 35th annual ACM symposium on applied computing* (pp. 181-189).

[89] Bakhshi, A., Noman, N., Chen, Z., Zamani, M., & Chalup, S. (2019, June). Fast automatic optimisation of CNN architectures for image classification using genetic algorithm. In *2019 IEEE congress on evolutionary computation (CEC)* (pp. 1283-1290). IEEE.

[90] Joshi, D., Mishra, V., Srivastav, H., & Goel, D. (2021). Progressive transfer learning approach for identifying the leaf type by optimizing network parameters. *Neural Processing Letters*, 53(5), 3653-3676.

[91] Liu, S., Zhang, H., & Jin, Y. (2022). A survey on surrogate-assisted efficient neural architecture search. *arXiv preprint arXiv:2206.01520*.

[92] Mishra, V., & Kane, L. (2023, March). Self-build Deep Convolutional Neural Network Architecture Using Evolutionary Algorithms. In *Proceedings of Fourth International Conference on Computer and Communication Technologies: IC3T 2022* (pp. 463-471). Singapore: Springer Nature Singapore.

[93] Joshi, D., Singh, T. P., & Sharma, G. (2022). Automatic surface crack detection using segmentation-based deep-learning approach. *Engineering Fracture Mechanics*, 268, 108467.

[94] Mishra, V., & Kane, L. (2023). A survey of designing convolutional neural network using evolutionary algorithms. *Artificial Intelligence Review*, 56(6), 5095-5132.

[95] Gavrilov, A. D., Jordache, A., Vasdani, M., & Deng, J. (2018). Preventing model overfitting and underfitting in convolutional neural networks. *International Journal of Software Science and Computational Intelligence (IJSSCI),* 10(4), 19-28.

[96] Dufourq, E., & Bassett, B. A. (2017, November). Eden: Evolutionary deep networks for efficient machine learning. In *2017 Pattern Recognition Association of South Africa and Robotics and Mechatronics (PRASA-RobMech)* (pp. 110-115). IEEE.

[97] Cai, H., Chen, T., Zhang, W., Yu, Y., & Wang, J. (2018, April). Efficient architecture search by network transformation. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 32, No. 1).

[98] Ma, B., Li, X., Xia, Y., & Zhang, Y. (2020). Autonomous deep learning: A genetic DCNN designer for image classification. *Neurocomputing*, 379, 152-161.

[99] Sabour, S., Frosst, N., & Hinton, G. E. (2017). Dynamic routing between capsules. *Advances in neural information processing systems,* 30.

[100] Zagoruyko, S., & Komodakis, N. (2016). Wide residual networks. *arXiv preprint arXiv:1605.07146.*

[101] Hu, J., Shen, L., & Sun, G. (2018). Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 7132-7141).

[102] Mishra, V., & Kane, L. (2023). An evolutionary framework for designing adaptive convolutional neural network. *Expert Systems with Applications*, 224, 120032.

[103] Huang, J., Xue, B., Sun, Y., Zhang, M., & Yen, G. G. (2022). Particle swarm optimization for compact neural architecture search for image classification. IEEE Transactions on Evolutionary Computation.

[104] Lawrence, T., Zhang, L., Lim, C. P., & Phillips, E. J. (2021). Particle swarm optimization for automatically evolving convolutional neural networks for image classification. IEEE access, 9, 14369-14386.

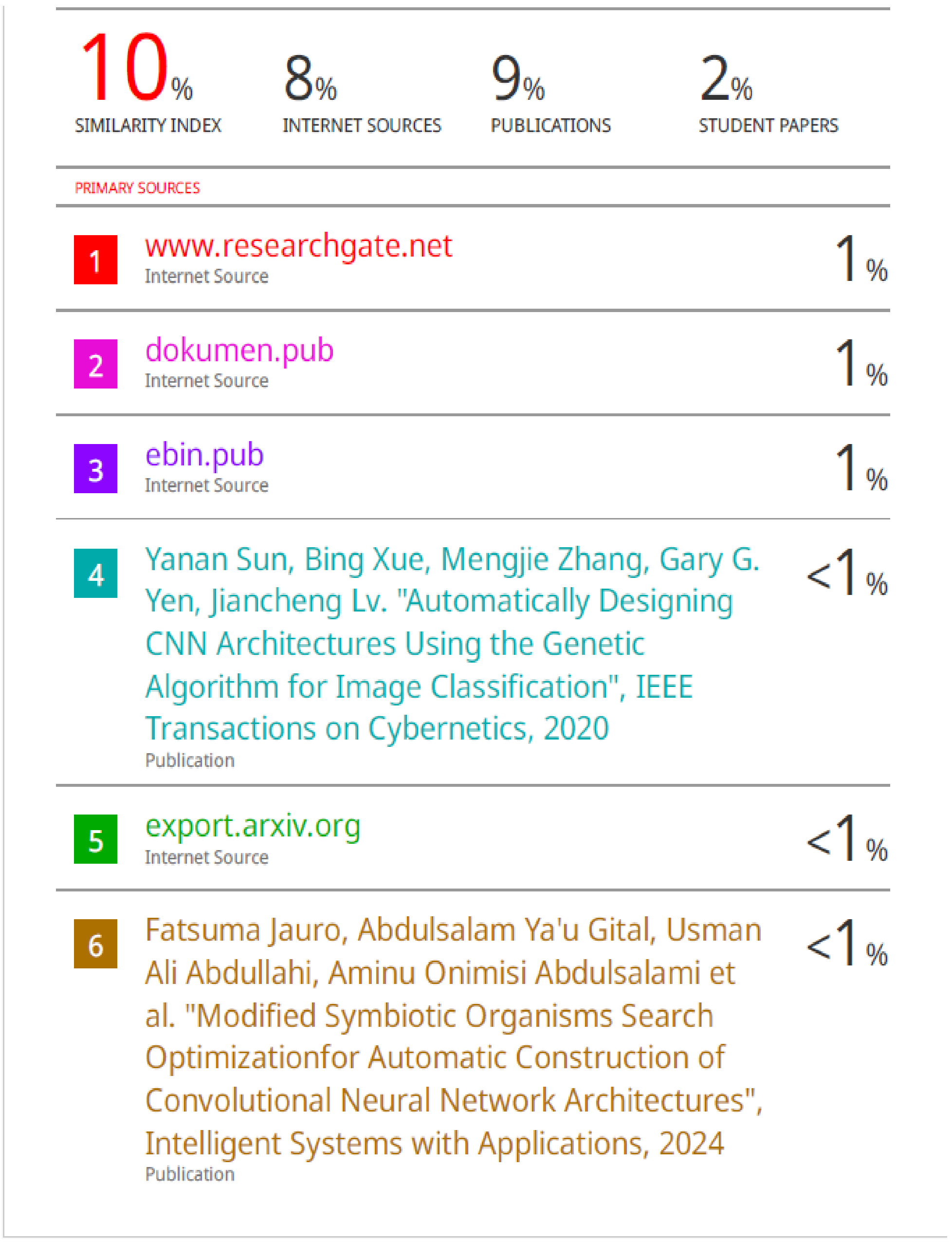
[105] Rajesh, C., & Kumar, S. (2022). An evolutionary block based network for medical image denoising using Differential Evolution. Applied Soft Computing, 121, 108776.

LIST OF PUBLICATIONS

1. Vidyanand Mishra, and Lalit Kane. "An evolutionary framework for designing adaptive convolutional neural network." Expert Systems with Applications 224 (2023): 120032.

2. Vidyanand Mishra, and Lalit Kane. "A survey of designing convolutional neural network using evolutionary algorithms." Artificial Intelligence Review 56.6 (2023): 5095-5132.

3. Vidyanand Mishra, and Lalit Kane. "Self-build Deep Convolutional Neural Network Architecture Using Evolutionary Algorithms." Proceedings of Fourth International Conference on Computer and Communication Technologies: IC3T 2022. Singapore: Springer Nature Singapore, 2023.



10%	8%	9%	2%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.researchgate.net Internet Source	1 %
2	dokumen.pub Internet Source	1 %
3	ebin.pub Internet Source	1 %
4	Yanan Sun, Bing Xue, Mengjie Zhang, Gary G. Yen, Jiancheng Lv. "Automatically Designing CNN Architectures Using the Genetic Algorithm for Image Classification", IEEE Transactions on Cybernetics, 2020 Publication	<1 %
5	export.arxiv.org Internet Source	<1 %
6	Fatsuma Jauro, Abdulsalam Ya'u Gital, Usman Ali Abdullahi, Aminu Onimisi Abdulsalami et al. "Modified Symbiotic Organisms Search Optimizationfor Automatic Construction of Convolutional Neural Network Architectures", Intelligent Systems with Applications, 2024 Publication	<1 %