Intelligent Systems with Applications 16 (2022) 200145

Contents lists available at ScienceDirect

Intelligent Systems with Applications

journal homepage: www.elsevier.com/locate/iswa

# A novel algorithm for identifying the optimal CNN Architectures regulated by Swarm Intelligence

Mangalraj [a,*], Mohit Pandya [b], Sayonee Dassani [c]

[a] School of Computer Science and Engineering, VIT-AP University, India
[b] Barclay Execution Services, India
[c] Optum Global Solutions, India

ARTICLE INFO

*Keywords:*
Deep learning
Swarm intelligence
Optimal architectures

ABSTRACT

The Emergence of deep learning architectures in recent years paves a pathway for developing several real-time applications. Deep learning architecture selection is time-consuming since it takes hours to get trained, few architectures take days to get training. To address the pitfall in selecting the architectures, we developed a novel algorithm that returns an optimal architecture to the user with the help of swarm intelligence. The parameters required to run the algorithm have been obtained dynamically from the input samples, so the returned architecture is reliable. The algorithm is tested based on qualitative metrics such as training accuracy and testing accuracy, and the algorithm is evaluated based on quantitative metrics such as convergence time and execution time. The algorithm is robust against any dataset, and the architecture generated by the algorithm yields better results.

## 1. Introduction

Computer vision is an interdisciplinary area of research, which integrates Artificial Intelligence, Mathematics as well as Analytic procedures to obtain insight from the image and video data. Computer vision has been leaving its footprints in almost every domain from agriculture to medicine. However, working with computer vision problems aims to address one of its primary issues, which is extracting information from vast structured and unstructured data. The advent of neural networks in artificial intelligence addressed the primary issue to a good extent by providing features as input and output has been generated with the self-tuning intelligence of the network. Identifying features from the vast data is not an easy task, which led the researchers to focus on the improvisation of neural networks which eventually led to the introduction of Convolutional Neural Networks (CNN's). The advent of CNN's made the feature identification automatic through the convolution process and in this process the network will identify insightful features that improves the process to a great extent.

The research in developing new architectures for CNN's has started long way from 5-layered LeNet (LeCun et al., 1998) to 200-layered DenseNet (Huang et al., 2017). Recent research into developing new layer types such as inception, residual and so on is in process. The research carried out in developing new architectures for CNN's emphasizing the importance of CNN's in Computer Vision and other domains. However, these architectures are completely dependent on the complexity of the data. If it is simple binary data, then even a 5-layered architecture works well with greater performance and for complex data, complex architectures are required. Identifying the best CNN architecture for a particular data set will be a tedious process from the available 100's of different complex architectures.

If a network is optimized for a particular data, then the ambiguity in selecting a particular CNN architecture from available architectures can be effectively addressed. In the proposed research work, we try to optimize a network architecture based on the data with the help of swarm intelligence. Swarm Intelligence (Ahmed & Glasgow, 2012) is a population-based model that helps in finding the most optimal solution from the available solution space. This intelligence has been reciprocated in our research by addressing the identification of an optimal architecture for CNN's based on the input data characteristics. A number of swarm intelligence algorithms have been developed in the current state of art, such as Ant Colony Optimizer (Dorigo et al., 2006), Ant Lion Optimizer (Mirjalili, 2015), Dragonfly Algorithm (Mirjalili, 2016), Genetic Algorithm (Whitley, 1994), Grey Wolf Optimizer (Mirjalili et al., 2014), Grasshopper Optimization Algorithm (Mirjalili et al., 2018), Particle Swarm Optimization (Kennedy & Eberhart, 1995), Artificial Bee Colony Algorithm (Karaboga, 2010) and so on.

* Corresponding author.
*E-mail address:* mangal86@gmail.com (Mangalraj).

https://doi.org/10.1016/j.iswa.2022.200145
Received 2 September 2021; Received in revised form 2 April 2022; Accepted 1 November 2022
Available online 5 November 2022

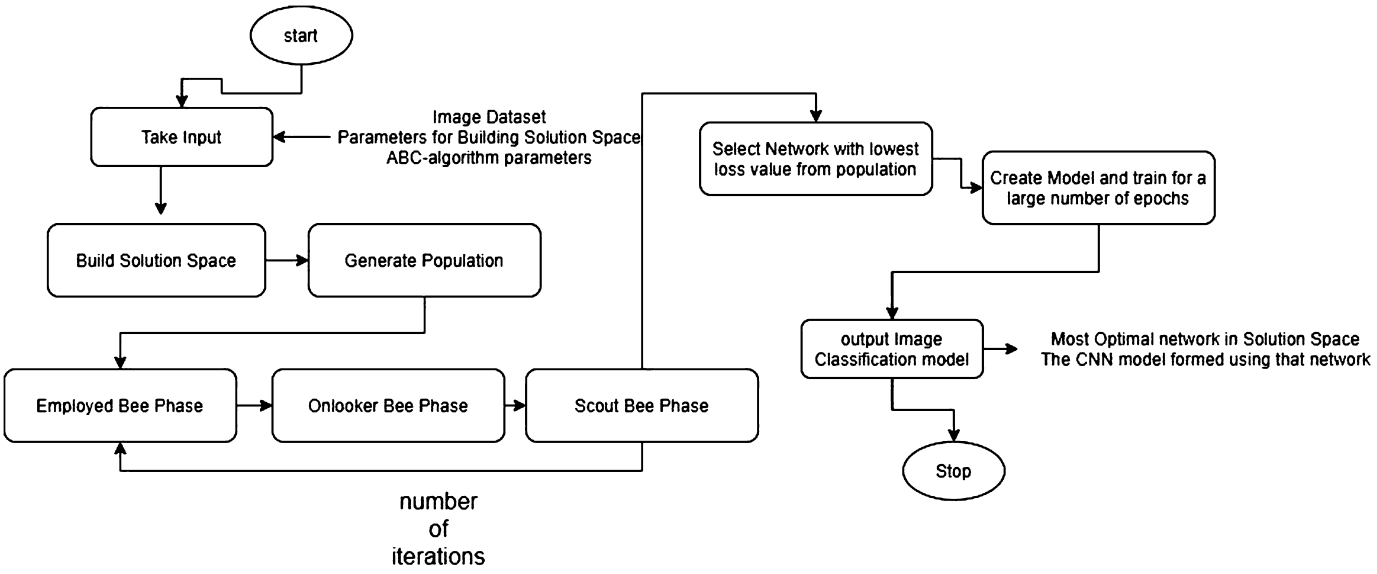


**Fig. 1.** Flowchart of the proposed method.

The Artificial Bee Colony (ABC) algorithm has been chosen for our proposed research, because of its converging speed and utilization of employing fewer control parameters as shown by Karaboga and Akay (2009). In the research work by Muthiah and Rajkumar (2014) they have proved that the ABC algorithm performs better than other algorithms due to its unique ability of how it executes local search by employee bees and universal search by onlooker and scout bees. The most important reason for us to select ABC for our proposed method is that, less number of control parameters that are to be set in this algorithm without sacrificing performance. The paper is organized as follows, the section 1 describes the introduction, the section 2 emphasizes on the algorithm, the section 3 shows the experiments, the section 4 elaborates the results and analysis, and the section 5 concludes the research work with remarks and scopes.

## 2. Optimizing CNN Architecture using ABC algorithm

In this section, we elaborate on how we used a swarm optimization technique to optimize the CNN architecture. In the proposed work, we considered different CNN architectures as the population (food source). The inputs to run the algorithm are layers (lower and upper bound), maximum kernel size, maximum output channel for the convolution layers, and maximum number of neurons in the fully connected layers. The other parameters which need to be given are iterations, swarm size and exiting criteria. The above-mentioned inputs and parameters need to be identified dynamically, which is an important aspect of the proposed work. The specified parameters and the inputs required for the algorithm are determined from the given input sample.

The proposed working model is depicted as a flow chart in Fig. 1. We can notice this from Fig. 1, our proposed model has three phases, the first phase of the model is generating the population, the second phase of the model is executing the ABC algorithm on the generated population and the third phase of the model is obtaining the optimal solution (CNN Architecture), if the obtained optimal solution not satisfying the desired accuracy the process will be continued with newer population.

Our model starts building the solution space using the obtained parameters. The solution space will generate a random population of CNN architectures which will be morphed into our solution space to obtain the population. Once the population is generated from the solution space, the three phases of the ABC algorithm come in, the employed bee phase, the onlooker bee phase and the scout bee phase. The three-phase pf ABC algorithm will be executed repeatedly until it reaches the exiting criteria. In these three phases the networks in the population will be trained on a small percentage of the dataset and for very little number of epochs to check their performance. On the basis of their performance (i.e. based on the loss value) the networks in the population will be removed, and modified with the intent to reach the most optimized model. So from among this population at the end of the 3-bee phases the one with the lowest loss is chosen. The optimal model obtained is then trained on the whole dataset and for more number of epochs to finally give an optimal and well-trained model. The three phases of the ABC algorithm and their corresponding Pseudocode have been discussed in the later sections.

### 2.1. Employed Bee Phase

In the Employed Bee Phase randomly generated food sources (CNN Architectures) are passed to a $generateNewFoodSrc()$ function, which processes the current architecture and the random one to produce a new architecture in the vicinity. After generating the new architecture, it is trained on a dataset to find the loss value with the help of an objective function. In ABC-CNN algorithm, the fitness function is calculated from the objective function as shown in equation (1). If the new architecture is more fit, then the current architecture is replaced with the new architecture. There is a value called $trial$, when the new architecture is selected, the $trial$ value is set to 0 else it is increased by the value 1. The trial value indicates, how many times the $generateNewFoodSrc()$ function fails to generate a better architecture.

$$fit_m(x_m) = \begin{Bmatrix} \frac{1}{1+f_m(x_m)} & \text{if} \quad f_m(x_m) \geq 0 \\ 1 + abs(f_m(x_m)) & \text{if} \quad f_m(x_m) < 0 \end{Bmatrix} \tag{1}$$

where $fit_m$ is the fitness of the $m_{th}$ particle/network in the population, $x_m$ is the $m_{th}$ particle/network and $f_m(x_m)$ is the objective function. In our algorithm, the objective function refers to the loss value of the network.

### 2.2. Onlooker Bee Phase

The probabilities for all the architectures are calculated using the $calculateProb()$ function. The fitness values are put in to the equation (2) to find the probabilities for the different CNN architectures in the population.

$$prob[i] = \frac{0.9 * fitness[i]}{max(fitness[])} + 0.1 \tag{2}$$

As shown in the algorithm ABC-CNN, we used two variables, $m$ and $n$ to keep track of the number of bees and architectures. In the algorithm a random number $r$ is selected. If $r$ is less than $prob[n]$ then the $n^{th}$ architecture is processed (i.e. a new architecture is generated, then fitness values are calculated) and its corresponding trial values are updated. The value of $m$ is incremented, indicating that we go to the next bee. After this, $n$ is incremented, indicating the next architecture. If the value of $r$ is greater than the probability, then only $n$ is incremented and the next architecture's probability is checked. If $n$ increases more than the population size $N$, then $n$ is set to zero. The significance of the particular architecture is decided by the total number of picks made by different onlooker bees.

### 2.3. Scout Bee Phase

The unemployed bees who choose their architecture randomly are called scouts. Architectures whose fitness values are not increased after a predetermined number of steps will be discarded. This number is tracked using the variable *trial* as shown in the algorithm ABC-CNN (Algorithm 1). The predetermined number of steps is decided by the input of ABC-CNN algorithm called as *limit*, which we call it abandonment criteria. In this phase the maximum trial value is found, and if that value is more than the limit, the architecture is discarded and a new random architecture is generated. While finding the optimal architecture, three cases need to be encountered.

1. No trial value is greater than limit, then the scout bee phase is skipped
2. If there are a few architectures whose trial value is greater than the limit, then one with maximum trial value is discarded
3. If there is more than one architecture whose trail value is greater, then the architecture needs to be discarded randomly to get an optimal one

### 2.4. Generating new architectures

In the algorithm GenerateNewArchitecture (Algorithm 2) a new architecture is generated using the currently selected architecture and a randomly selected one from the population. In this algorithm, determining the convolution layer is carried out and the possible changes will be made on the same to obtain a new architecture. In particular, changes have been carried out on the output channels of the convolution layers. Equation (3) is used to calculate the new channel number for the selected convolution layer.

$$\upsilon_m = x_m + \phi_m(x_m - x_k) \tag{3}$$

where $m$ is the index of the current network selected in the population, $k$ is the index of the random network selected in the population, $\upsilon$ is the new network generated, $x$ refers to the original networks and $\phi$ refers to the random value.

The Algorithm GenerateNewArchitecture randomly selects a layer and the number of output channels is recorded. The selected layer must be a convolution layer; if not the next layer will be checked, the process repeats until a convolution layer is found. The output channel is assigned to the particular layer selected before and the new architecture is returned.

## 3. Experiment details

### 3.1. Setup

The research work has been carried out using the Virtual Machine (VM) of Google Compute Engine platform. The specifications of the VM are, *n1-standard-8 (8 vCPUs, 30 GB memory)* as the CPU and the GPU was a *1 x NVIDIA Tesla T4* having 16 GB of memory which makes the algorithm to execute without any hiccups. After successfully executing

**Algorithm 1:** ABC-CNN Algorithm.

**Input :** Number of Iteration ($iter$), population size ($N$), min number of layers ($l_{min}$), maximum number of layers ($l_{max}$), max output channel for convolution layer ($max_{conv}$), max output channel for fully connected layer ($max_{FC}$), limit ($limit$), epochs, dataset
**Output:** Architecture and metrics of the best Solution and Best Discarded Solution

```
pop ← Population(N, l_min, l_max, max_conv, max_FC)
for i ← 1 to iter do
    Employed Bee Phase
    for foodSrc in pop do
        x ← Random (0,N);
        GenerateNewFoodSrc (foodSrc.layer ,pop[x].layers);
        fit, fit_new ←CalculateFitness ();
        if fit < fit_new then
            foodSrc ← NewFoodsrc;
            trial = 0;
        else trial++;
        ;
    prob ← calculateProb ();
    Onlooker Bee Phase
    m = 0;
    n = 0;
    while m < N do
        r ← Random (0,1);
        if r < prob[n] then
            GenerateNewFoodSrc (pop[n].layer ,pop[Random (0,N)].layers);
            fit, fit_new ←CalculateFitness ();
            if fit < fit_new then
                foodSrc ← NewFoodsrc;
                trial = 0;
            else trial++;
            ;
            m++;
        n++;
        if n == N then
            n = 0;
    Scout Bee Phase
    max_trial ← max(trial);
    maxTrial_pos ← maxIndex(trial);
    if max_trial >= limit then
        if BestDiscardedSolnisempty then
            BestDiscardedSoln ← pop[maxTrial_pos];
        else
            if BestDiscardedSoln[loss] > pop[maxTrial_pos][loss] then
                BestDiscardedSoln ← pop[maxTrial_pos];
We can now select the BestArchitecture from the population or the BestDisacrdedSoln, which ever is having the less loss value
BestSolution ← minLoss (pop);
BestDiscardedSoln
```

**Algorithm 2:** GenerateNewArchitecture.

**Input :** Current Architecture ($p1$), Randomly selected Architecture ($p2$)
**Output:** Newly found Architecture

```
p1_fc ← Index of first fc layer (p1);
p2_fc ← Index of first fc layer (p2);
min_fc ← Minimum (p1_fc, p2_fc);
pos_conv ← Random (0,min_fc);
while p1[pos_conv] and p2[pos_conv] are not conv layers do
    pos_conv++;
    if pos_conv > min_fc then
        pos_conv = 0
out1 = p1[pos_conv][outChannel];
out2 = p2[pos_conv][outChannel];
r ← Random (0,1);
new_out ← out1 + r * (out1 − out2);
p1[pos_conv][outChannel] = newout
return p1;
```

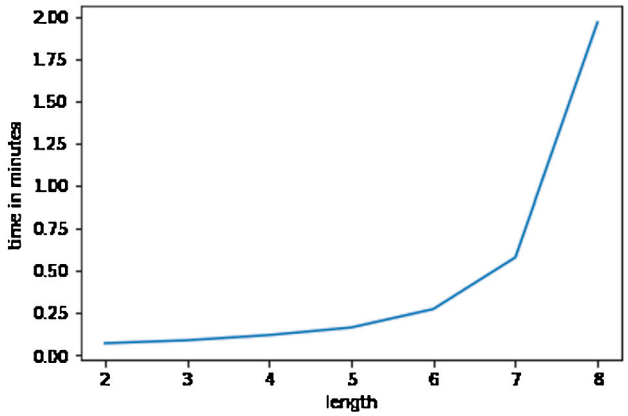


**Fig. 2.** Time taken to train vs length of network.

the program on a small dataset on the local machine, the code repository was threaded to the Github; the clones will be further processed on the Virtual Machine to execute the optimal architecture in the complete dataset for validation purposes.

### 3.2. Dataset

The research has been carried out on two datasets having different complexities. The first dataset is MNIST which is a simple binary handwritten dataset having 60000 training and 10000 testing samples with 10 classes. At the same time, we used CIFAR 10 dataset, which is quite complex and has 60000 color images with 10 different classes. Since CNN architectures are completely dependent on the nature of the data, we tried to compare our proposed method with these different datasets.

### 3.3. Parameters – assumption and estimation

#### 3.3.1. Dynamic selection of parameters

The parameters need to be provided to the proposed model in a dynamic way. The initial set of parameters govern the boundaries of CNN Architecture, which create inputs randomly (i.e. Set of CNN Architectures) for the proposed method. The length of the CNN architectures are presumed for lower bound as 3 and for the upper bound as 10. The lower and upper bound of the CNN architecture restricts the search space for obtaining an optimal CNN architecture for any given set of sample. The lower bound is considered as 3, because the CNNs will have at least one convolution layer, one pooling layer and a fully connected layer. The upper bound is chosen as 10, because when the length of the architecture increases the time complexity increases exponentially as depicted in Fig. 2.

The graph shown in Fig. 2 is plotted while testing the algorithm for training 10% of the original data sample for 10 epochs. It is evident from the graph that time complexity increases exponentially to a greater peak when the length of the architecture increases.

#### 3.3.2. Selection of pooling layer

While selecting layers for the CNN architecture, we have prioritized pooling layers. The main reason for selecting the pooling layer is that the dimension of the architecture decreases way too much till it reaches its end. Increasing the number of pooling layers naturally simplifies the learning curve. However, we have dynamically selected the number of pooling layers by the $prob_{Pool}$ function. Example of $prob_{Pool}$ function, if the number of pooling layer is 4, then the length and breadth of CNN will be decreased by $(2 \times 2, 28/2 = 14/2 = 7/2 = 3.5 - > 4/2 = 2)$ by 2 times.

**Table 1**
ABC-CNN input parameters.

| Solution space parameters | Value |
|---|---|
| Minimum length of the network | 3 |
| Maximum length of the network | 10 |
| Minimum number of output channels | 3 |
| Maximum Number of output channels | 256 |
| Minimum number of neurons in FC layer | 1 |
| Maximum number of neurons in FC layer | 300 |
| Minimum kernel size of Conv layer | $3 \times 3$ |
| Maximum kernel size of Conv layer | $7 \times 7$ |
| **ABC algorithm parameter** | **Values** |
| Number of Iteration | 10 |
| Population Size | 10 |
| Limit | 1 |
| **CNN training parameters** | **Values** |
| Epochs for finding objective function | 1 |
| Epochs for training optimal network | 50 |
| Batch Normalize layer output | *Yes* |
| Dropout rate | 0.5 |

#### 3.3.3. Default parameters used to get optimal solution

We selected a few default parameters to run the ABC-CNN algorithm. These default parameters have been selected to identify the optimal architecture in context with the small set of input samples.

#### 3.3.4. Fitness function

The fitness function has to be determined while training the network; in our case we have considered *Epochs for finding objective function*, since there will be a lot of instances of calling the fitness function, this value is set to 1. After the ABC algorithm is over and the optimal network is selected, it is trained for *Epochs for training optimal network* times so as to get the best results with the most optimal network.

#### 3.3.5. Analysis of architectures and accuracies

We have used a small set of input sample to identify the optimal architecture, which is shown in Fig. 3. From the figure it is visible that even for an architecture, which has a length less than 10 performs well. From the graph it is evident that, we will be able to identify the optimal network, with minimal parameter selection (Table 1). Further results and analysis have been carried out in the later section.

## 4. Results

### 4.1. Experiments

The experiments have been conducted on two different datasets of different complexities (MNIST and CIFAR 10). The experiments have been conducted on these datasets to perform a comparative study of the proposed algorithm. The initial population of CNN architectures and their corresponding training accuracies have been provided in Table 2. Table 2 has been generated while training the initial population (Different CNN Architectures) on CIFAR-10 dataset. From Table 2, it is evident that the initial training accuracies on the population are less than 50%. From the initial population, the best network is identified after the completion of CNN-ABC algorithm.

### 4.2. Results

The network architecture is represented in the following way. Each layer is represented as, $LayerType^{kernelsize}_{outputchannel}$ and the types of layers used to generate network architecture are Convolutional ($conv$), Fully Connected ($FC$) and Pooling layer ($Pool$). The algorithm only needs to run once and it will produce 2 optimal networks, especially the best network from the population and the other one is the best discarded network. Here “discarded” does not mean that is not good and of no use; it is called discarded because since it was such a good network it was stopping the algorithm from exploring the solution space, so it is

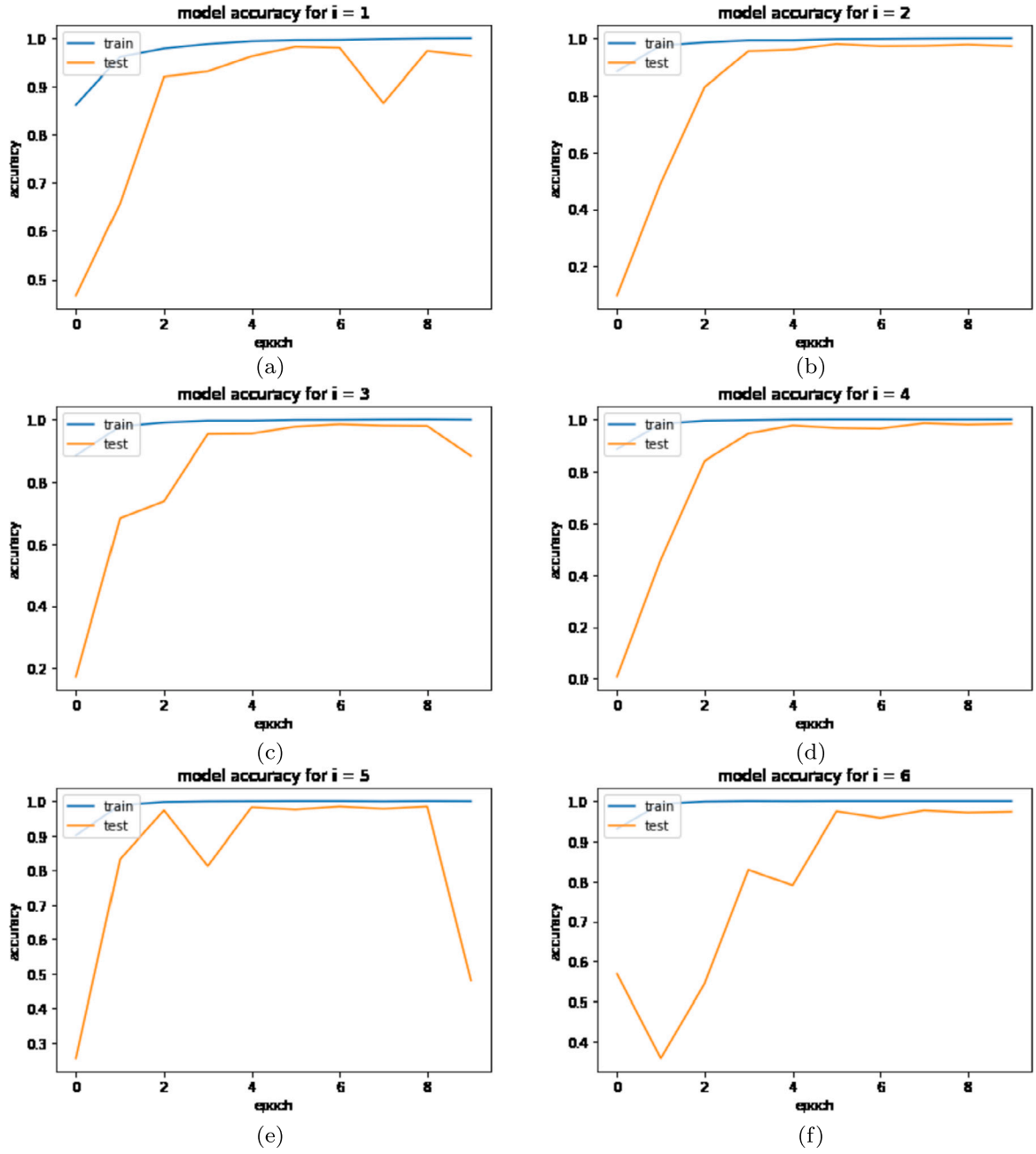


**Fig. 3.** Test and train accuracies for different lengths, where number of conv and pool layers equal to i + 1.

discarded from the solution space and saved as the best discarded network. At the end we give both networks as output, but the user can choose any one with higher accuracy. So at every Run the user will get 2 optimal networks named as best network and best discarded, he has the freedom to choose the best out of 2. If the user is not satisfied with the optimal results he can go for the second run.

Every run of the algorithm yields the Best Network with its corresponding Train and Test Accuracy as well as train and test loss similarly the algorithm yields the best discarded network with its corresponding Train and Test Accuracy as well as train and test loss. We evaluated our algorithm by generating the best architectures in three runs. We have used the minimal sample to identify the best architecture through the algorithm and the optimal network is evaluated in the complete dataset to show the performance of the optimal architecture. The results of the optimal networks executed in the complete datasets and their corresponding accuracies and losses have been detailed in Table 3.

From Table 2 and from 3, it is evident that our CNN-ABC algorithm converges well in identifying the optimal architectures with training accuracies above 90% for the complete dataset.

### 4.3. Analysis

In Table 2 the architectures in the population produced poor results, which were below 50%. After converging the population with the help of proposed CNN-ABC Algorithm, the results were improved with an average training accuracy of 90% and the optimal network has been generated based on the qualitative parameters.

#### 4.3.1. On MNIST

When the complete dataset MNIST is executed on the optimal network $conv^{5}_{206} \rightarrow conv^{4}_{135} \rightarrow conv^{5}_{143} \rightarrow FC_{10}$ the training accuracy was .9994 and the testing accuracy as 0.9945. On the other hand the best

**Table 2**
Training accuracies for all the particles (networks) in the population for CIFAR-10 dataset.

| | | Iterations | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Population | Run 1 | 0.3626 | 0.355 | 0.3486 | 0.349 | 0.349 | 0.349 | 0.349 | 0.3488 | 0.3602 | 0.3602 |
| | | 0.345 | 0.345 | 0.345 | 0.3486 | 0.3486 | 0.3486 | 0.3486 | 0.3486 | 0.3628 | 0.3628 |
| | | 0.1756 | 0.1756 | 0.3672 | 0.3672 | 0.3672 | 0.3766 | 0.3766 | 0.3692 | 0.3692 | 0.3692 |
| | | 0.2884 | 0.2884 | 0.2884 | 0.2884 | 0.2884 | 0.234 | 0.234 | 0.234 | 0.234 | 0.234 |
| | | 0.3532 | 0.3532 | 0.3532 | 0.3532 | 0.3532 | 0.361 | 0.363 | 0.363 | 0.363 | 0.363 |
| | | 0.2572 | 0.2764 | 0.2764 | 0.2764 | 0.2788 | 0.2788 | 0.2788 | 0.2788 | 0.2788 | 0.2788 |
| | | 0.3302 | 0.3302 | 0.3302 | 0.3302 | 0.3302 | 0.3302 | 0.3614 | 0.3614 | 0.3614 | 0.3614 |
| | | 0.3702 | 0.3702 | 0.3702 | 0.3672 | 0.3672 | 0.3672 | 0.3672 | 0.3672 | 0.3672 | 0.3374 |
| | | 0.2684 | 0.2684 | 0.2684 | 0.2684 | 0.3558 | 0.3558 | 0.3558 | 0.3558 | 0.3558 | 0.3558 |
| | | 0.1212 | 0.1212 | 0.1212 | 0.1212 | 0.1262 | 0.1262 | 0.1264 | 0.1264 | 0.1264 | 0.1264 |
| | Run 2 | 0.3584 | 0.3714 | 0.3832 | 0.3832 | 0.3832 | 0.3832 | 0.3576 | 0.3702 | 0.3702 | 0.3702 |
| | | 0.2872 | 0.2872 | 0.2872 | 0.2872 | 0.377 | 0.377 | 0.377 | 0.3106 | 0.315 | 0.315 |
| | | 0.3356 | 0.3356 | 0.3356 | 0.114 | 0.114 | 0.114 | 0.112 | 0.115 | 0.1132 | 0.1132 |
| | | 0.2322 | 0.2322 | 0.2322 | 0.2322 | 0.274 | 0.274 | 0.274 | 0.274 | 0.274 | 0.2618 |
| | | 0.2658 | 0.2658 | 0.3532 | 0.3532 | 0.3658 | 0.3658 | 0.3658 | 0.3658 | 0.3658 | 0.3658 |
| | | 0.372 | 0.3792 | 0.3792 | 0.3792 | 0.3772 | 0.3772 | 0.3744 | 0.3744 | 0.3744 | 0.3744 |
| | | 0.2664 | 0.2664 | 0.2664 | 0.2664 | 0.2664 | 0.2764 | 0.2764 | 0.2764 | 0.2764 | 0.2764 |
| | | 0.3704 | 0.3704 | 0.377 | 0.377 | 0.377 | 0.377 | 0.377 | 0.377 | 0.353 | 0.353 |
| | | 0.305 | 0.305 | 0.3088 | 0.3298 | 0.3298 | 0.3298 | 0.3212 | 0.3212 | 0.3212 | 0.3212 |
| | | 0.3102 | 0.3102 | 0.3102 | 0.3026 | 0.3034 | 0.3034 | 0.3034 | 0.3024 | 0.3 | 0.3 |
| | Run 3 | 0.2402 | 0.236 | 0.2464 | 0.2464 | 0.2464 | 0.3822 | 0.3822 | 0.3786 | 0.3786 | 0.3798 |
| | | 0.3456 | 0.364 | 0.3738 | 0.3738 | 0.3738 | 0.3738 | 0.3184 | 0.3184 | 0.3132 | 0.3314 |
| | | 0.359 | 0.359 | 0.359 | 0.3586 | 0.3748 | 0.3748 | 0.3748 | 0.3748 | 0.3602 | 0.3602 |
| | | 0.365 | 0.3726 | 0.3726 | 0.3726 | 0.3726 | 0.3706 | 0.3706 | 0.3706 | 0.3706 | 0.3706 |
| | | 0.333 | 0.3332 | 0.3332 | 0.3332 | 0.3332 | 0.3332 | 0.3332 | 0.099 | 0.1066 | 0.1084 |
| | | 0.3614 | 0.3718 | 0.3718 | 0.3718 | 0.3718 | 0.3718 | 0.3718 | 0.3718 | 0.3466 | 0.3466 |
| | | 0.3506 | 0.3604 | 0.3604 | 0.3604 | 0.3558 | 0.3558 | 0.3558 | 0.3558 | 0.3558 | 0.3606 |
| | | 0.2822 | 0.2822 | 0.1218 | 0.1248 | 0.1102 | 0.1102 | 0.1074 | 0.1074 | 0.1074 | 0.1074 |
| | | 0.2072 | 0.2174 | 0.2174 | 0.2174 | 0.218 | 0.218 | 0.218 | 0.218 | 0.2104 | 0.2104 |
| | | 0.364 | 0.3508 | 0.3508 | 0.3508 | 0.3508 | 0.3508 | 0.3508 | 0.3508 | 0.3508 | 0.3576 |

**Table 3**
Comparative Results of ABC-CNN Algorithm in MNIST and CIFAR 10 Dataset.

| Runs | Parameters | MNIST | CIFAR10 |
|---|---|---|---|
| Run 1 | Best Network | $conv^5_{206} \rightarrow conv^4_{135} \rightarrow conv^5_{143} \rightarrow FC_{10}$ | $conv^5_{195} \rightarrow conv^3_{140} \rightarrow avg-pool^2_{-1} \rightarrow FC_{10}$ |
| | Train loss | 0.0020 | 0.6221 |
| | Train Accuracy | 0.9994 | 0.7856 |
| | Test loss | 0.0193 | 0.8601 |
| | Test Accuracy | 0.9945 | 0.7067 |
| | Best Discarded Network | $conv^5_{219} \rightarrow conv^5_{245} \rightarrow FC_{10}$ | $conv^6_{114} \rightarrow conv^4_{143} \rightarrow FC_{10}$ |
| | Train loss | 0.0019 | 0.3827 |
| | Train Accuracy | 0.9995 | 0.8682 |
| | Test loss | 0.0222 | 1.0353 |
| | Test Accuracy | 0.9937 | 0.6699 |
| | Time taken(s) | 2087.97 | 2112.29 |
| Run 2 | Best Network | $conv^3_{211} \rightarrow conv^6_{200} \rightarrow conv^3_{196} \rightarrow FC_{10}$ | $conv^3_{109} \rightarrow conv^4_{251} \rightarrow FC_{10}$ |
| | Train loss | 0.0024 | 0.2635 |
| | Train Accuracy | 0.9995 | 0.9109 |
| | Test loss | 0.0201 | 1.0696 |
| | Test Accuracy | 0.9945 | 0.6650 |
| | Best Discarded Network | $conv^4_{235} \rightarrow conv^5_{157} \rightarrow conv^5_{116} \rightarrow conv^5_{146} \rightarrow FC_{10}$ | $conv^4_{246} \rightarrow conv^5_{235} \rightarrow conv^6_{229} \rightarrow FC_{10}$ |
| | Train loss | 0.0020 | 0.1708 |
| | Train Accuracy | 0.9993 | 0.9460 |
| | Test loss | **0.0181** | 0.7128 |
| | Test Accuracy | 0.9944 | 0.7831 |
| | Time taken(s) | 2356.89 | 2822.60 |
| Run 3 | Best Network | $conv^4_{250} \rightarrow conv^6_{180} \rightarrow conv^3_{191} \rightarrow FC_{10}$ | $conv^5_{198} \rightarrow conv^4_{81} \rightarrow FC_{10}$ |
| | Train loss | 0.0023 | 0.44340 |
| | Train Accuracy | 0.9993 | 0.8456 |
| | Test loss | 0.0205 | 1.0241 |
| | Test Accuracy | 0.9944 | 0.6625 |
| | Best Discarded Network | $conv^3_{119} \rightarrow conv^3_{73} \rightarrow conv^3_{173} \rightarrow conv^6_{226} \rightarrow FC_{10}$ | $conv^4_{246} \rightarrow conv^5_{235} \rightarrow conv^6_{229} \rightarrow FC_{10}$ |
| | Train loss | **0.0019** | 0.1708 |
| | Train Accuracy | **0.9996** | 0.9460 |
| | Test loss | 0.02056 | 0.7128 |
| | Test Accuracy | **0.9947** | 0.7831 |
| | Time taken(s) | 1916.03 | 2822.60 |

discarded network $conv^5_{219} \rightarrow conv^5_{245} \rightarrow FC_{10}$ yields a training accuracy of .9995 and the test accuracy as .9937. At the first run itself, for the Complete MNIST dataset we obtained the best accuracies in both the optimal architectures. To validate further more, we carried out two more runs, however in the other two runs the algorithm yielded 4 more optimal solutions and the results are provided in Table 3. After the 2nd run the training accuracies are 0.9995 and 0.9993 similarly after the 3rd run the training accuracies are 0.9993 and 0.9996. The testing accuracies after the 2nd run are 0.9945 and 0.9944, similarly after the 3rd run the testing accuracies are 0.9944 and 0.9947. If the notice care-

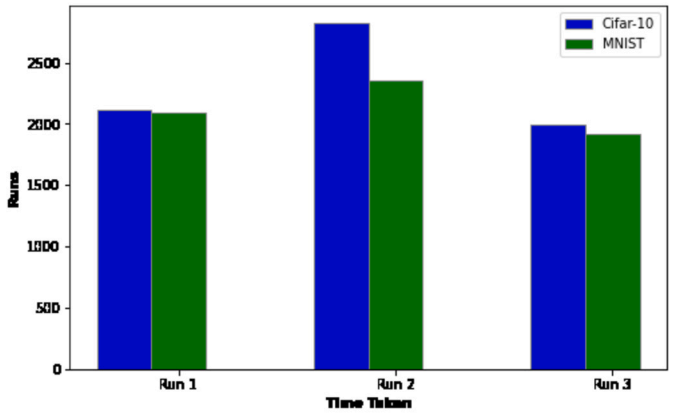


**Fig. 4.** Time taken by algorithm to run.

fully the architectures generated in the 2nd and 3rd run yielded similar training and testing accuracies, which furthermore proves that the algorithm has converged at its best solutions. More number of runs will yield more number of optimal architectures for the user, the freedom of choosing the optimal architecture is based on the user needs such as time complexity or qualitative metrics.

#### 4.3.2. On CIFAR10

In the CIFAR dataset in the first run we obtained best trained network as $conv^{5}_{195} \rightarrow conv^{3}_{140} \rightarrow avg-pool^{2}_{-1} \rightarrow FC_{10}$ and it's corresponding training and test accuracies are 0.780 and 0.706 respectively. At the same run 1 the obtained best discarded network $conv^{6}_{114} \rightarrow conv^{4}_{143} \rightarrow FC_{10}$ yields the training and testing accuracies as 0.868 and 0.669. At this point the user can choose any one network or he can go for one more Run until he gets his desired output. From Table 3, the Run 2 results have improved with new other 2 optimal architectures. The best trained network $conv^{3}_{211} \rightarrow conv^{6}_{200} \rightarrow conv^{3}_{196} \rightarrow FC_{10}$ yields the training and testing accuracies as 0.9109 and 0.6650 respectively. The best discarded network obtained in the run 2 is $conv^{4}_{246} \rightarrow conv^{5}_{235} \rightarrow conv^{6}_{229} \rightarrow FC_{10}$, which returns the training accuracy as 0.9460 and testing accuracy as 0.7831. The Run 2 yields better optimal network than run 1 but we tried again to go for Run 3. In Run 3 the best training network $conv^{5}_{198} \rightarrow conv^{4}_{81} \rightarrow FC_{10}$ produced a training accuracy of 0.8456 which is less than the training accuracy produced by the best training network of Run 2. Similarly the best discarded network generated in both the runs are same (i.e. $conv^{4}_{246} \rightarrow conv^{5}_{235} \rightarrow conv^{6}_{229} \rightarrow FC_{10}$), which proves that further convergence of newer solutions is not possible from the given population. If the user needs to improve the accuracies, he can feed the obtained newer solutions to the population and the user can restart the whole process. The algorithm doesn't guarantee the user for the best solution, but it guarantees for the optimal one in a shorter time span.

#### 4.3.3. On execution time

We have carried out an inbound analysis on the execution time of the algorithm. The execution time comparison is depicted in Fig. 4. After the completion of identifying the best trained network and best discarded network, the complete datasets have been executed on the networks for validation purpose and the qualitative metrics based analysis has been carried out in the former section.

The execution time taken for the MNIST dataset is less than CIFAR dataset. The reason for the time lag is that the CIFAR is a color image dataset and MNIST is a grayscale image dataset so the variation occurs due to the band difference. The MNIST takes execution time in run 1, run 2, and run 3 as 2087.97, 2356.89 and 1916.03 Seconds, respectively. In the run 2, MNIST has taken much time to execution because of the best discarded architecture length, which we already discussed in the section 3.3.1. When the length of the architecture increases, the time taken for execution increases exponentially. The CIFAR 10 takes an execution time in run 1, run 2 and run 3 as 2112.29, 2822.60 and 2012.54 Seconds, respectively. The execution time of run 3 has been decreased, which eventually proves that the convergence speed of the algorithm increases after multiple runs. When the best discarded network is the same in different runs, which means that a further more optimal solution cannot be generated from the algorithm. For further exploration of optimal solutions, the population needs to be changed with the obtained optimal solutions and with new architectures.

## 5. Conclusion

In the research work, we proposed a novel algorithm to find an optimal CNN architecture for a particular dataset. The proposed algorithm is robust against any kind of dataset; we have evaluated the algorithm against two different datasets with different complexities. The proposed ABC-CNN is fast enough to identify the optimal architecture with respect to the input dataset. The quantitative and qualitative evaluations prove that our proposed algorithm performs well in identifying the optimal architecture. The user has been given the freedom of identifying multiple optimal architectures, which is an added advantage by the algorithm. The algorithm does not guarantee the best architecture, but it guarantees the optimal one within a short span of time.

## CRediT authorship contribution statement

**Mangalraj:** Conception and design of study, Revising the manuscript critically for important intellectual content, Approval of the version of the manuscript to be published. **Mohit Pandiya:** Analysis and/or interpretation of data, Drafting the manuscript, Approval of the version of the manuscript to be published. **Sayonee Dassani:** Acquisition of data, Approval of the version of the manuscript to be published.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## References

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE*, *86*(11), 2278–2324.

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).

Ahmed, H., & Glasgow, J. (2012). *Swarm intelligence: concepts, models and applications*. School of Computing, Queens University. Technical Report.

Dorigo, M., Birattari, M., & Stutzle, T. (2006). Ant colony optimization. *IEEE Comput. Intell. Mag.*, *1*(4), 28–39.

Mirjalili, S. (2015). The ant lion optimizer. *Adv. Eng. Softw.*, *83*, 80–98.

Mirjalili, S. (2016). Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems. *Neural Comput. Appl.*, *27*(4), 1053–1073.

Whitley, D. (1994). A genetic algorithm tutorial. *Stat. Comput.*, *4*(2), 65–85.

Mirjalili, S., Mirjalili, S. M., & Lewis, A. (2014). Grey wolf optimizer. *Adv. Eng. Softw.*, *69*, 46–61.

Mirjalili, S. Z., Mirjalili, S., Saremi, S., Faris, H., & Aljarah, I. (2018). Grasshopper optimization algorithm for multi-objective optimization problems. *Appl. Intell.*, *48*(4), 805–820.

Kennedy, J., & Eberhart, R. (1995). *Particle swarm optimizationProceedings of ICNN'95 – international conference on neural networks, Vol. 4*. IEEE (pp. 1942–1948).

Karaboga, D. (2010). Artificial bee colony algorithm. *Scholarpedia*, *5*(3), 6915.

Karaboga, D., & Akay, B. (2009). A comparative study of artificial bee colony algorithm. *Appl. Math. Comput.*, *214*(1), 108–132.

Muthiah, A., & Rajkumar, R. (2014). A comparison of artificial bee colony algorithm and genetic algorithm to minimize the makespan for job shop scheduling. *Proc. Eng.*, *97*, 1745–1754.