mathematics

MDPI

Article

Automated CNN Architectural Design: A Simple and Efficient Methodology for Computer Vision Tasks

Ali Al Bataineh [1,*], Devinder Kaur [2], Mahmood Al-khassaweneh [3,4] and Esraa Al-sharoa [5]

[1] Department of Electrical and Computer Engineering, Norwich University, Northfield, VT 05663, USA
[2] Department of Electrical Engineering and Computer Science, University of Toledo, Toledo, OH 43606, USA
[3] Engineering, Computing and Mathematical Sciences, Lewis University, Romeoville, IL 60446, USA
[4] Computer Engineering Department, Yarmouk University, Irbid 21163, Jordan
[5] Electrical Engineering Department, Jordan University of Science and Technology, Irbid 22110, Jordan
* Correspondence: aalbatai@norwich.edu

Abstract: Convolutional neural networks (CNN) have transformed the field of computer vision by enabling the automatic extraction of features, obviating the need for manual feature engineering. Despite their success, identifying an optimal architecture for a particular task can be a time-consuming and challenging process due to the vast space of possible network designs. To address this, we propose a novel neural architecture search (NAS) framework that utilizes the clonal selection algorithm (CSA) to automatically design high-quality CNN architectures for image classification problems. Our approach uses an integer vector representation to encode CNN architectures and hyperparameters, combined with a truncated Gaussian mutation scheme that enables efficient exploration of the search space. We evaluated the proposed method on six challenging EMNIST benchmark datasets for handwritten digit recognition, and our results demonstrate that it outperforms nearly all existing approaches. In addition, our approach produces state-of-the-art performance while having fewer trainable parameters than other methods, making it low-cost, simple, and reusable for application to multiple datasets.

Keywords: clonal selection algorithm (CSA); computer vision; convolutional neural networks (CNN); deep learning EMNIST; neural architecture search (NAS)

MSC: 68T45

Citation: Al Bataineh, A.; Kaur, D.; Al-khassaweneh, M.; Al-sharoa, E. Automated CNN Architectural Design: A Simple and Efficient Methodology for Computer Vision Tasks. *Mathematics* **2023**, *11*, 1141. https://doi.org/10.3390/math11051141

Academic Editor: Junlin Hu

Received: 30 January 2023
Revised: 18 February 2023
Accepted: 23 February 2023
Published: 24 February 2023

1. Introduction

Deep learning has revolutionized the field of computer vision by enabling automatic feature extraction, eliminating the need for manual feature engineering [1,2]. Recent advancements in deep learning have been driven by three key factors: the availability of large datasets, high-performance computing resources, and new and improved algorithms, such as CNN [3] and have made previously impossible smart applications widely available, from image recognition to autonomous driving [1]. Despite their success, designing the optimal CNN architecture remains a challenge, as it involves critical design decisions that affect accuracy and speed, such as layer organization, hyperparameters, and activation functions [4]. While some efforts have been made to automate the discovery of the best CNN architecture, many modern architectures still rely on human expertise and intuition gained from experimentation. Fortunately, nature-inspired optimization algorithms offer an alternative approach to automated CNN architecture design [5].

In this paper, we propose a new and efficient neural architecture search (NAS) framework based on CSA for automatically designing high-quality convolutional neural network (CNN) architectures for image classification tasks. CSA is a nature-inspired optimization algorithm that mimics the immune system's clonal selection process to generate high-affinity antibodies that recognize antigens [6,7]. In the proposed framework, we use an integer

vector representation to encode CNN architectures and hyperparameters, combined with a truncated Gaussian mutation scheme that enables efficient exploration of the search space. Our approach offers a promising and efficient alternative to traditional manual design methods for deep learning architectures, leveraging the power of nature-inspired optimization algorithms to overcome the challenge of finding the optimal CNN architecture. We evaluate our methodology on challenging datasets such as EMNIST, which contains both digits and letters, and achieve cutting-edge performance in terms of accuracy and model complexity.

The contributions of our research are as follows:

1. We propose a new and efficient NAS framework based on CSA for automatically designing high-quality CNN architectures for image classification tasks.
2. We introduce an integer encoding scheme for encoding hyperparameters and the layers of the CNN architecture, which enables efficient implementation of different mutation types.
3. We evaluate our methodology on challenging datasets such as EMNIST, which contains both digits and letters, and achieved cutting-edge performance in terms of accuracy and model complexity.
4. Our proposed method outperforms nearly all existing approaches, while having fewer trainable parameters than other methods, making it low-cost, simple, and reusable for application to multiple datasets.
5. Our approach provides a promising and efficient alternative to traditional manual design methods for deep learning architectures, leveraging the power of nature-inspired optimization algorithms to overcome the challenge of finding the optimal CNN architecture.

This paper is organized as follows: In Section 2, we provide a comprehensive review of existing NAS methods for designing CNN architectures, highlight their limitations, and explain the unique contributions of our proposed NAS framework. Section 3 provides a brief background on CNN and presents the CSA. In Section 4, we detail the implementation of the proposed NAS-based CSA approach for automatically designing CNN architectures. The datasets used and the experimental setup, including hyperparameters and evaluation metrics, are described in Section 5. In Section 6, we present the results of our experiments, including a detailed discussion of the findings. Finally, Section 7 summarizes the main findings of the research, discusses their implications, and proposes potential directions for future research.

2. Related Work

CNN models have been highly successful in a variety of applications due to their ability to automate the feature engineering process by learning "hierarchical feature extracts" from data [8]. While human experts have traditionally designed successful network architectures, there is a growing interest in the use of NAS techniques to automatically learn high-performance architectures. Some commonly used NAS algorithms for evolving network architectures are grid search [9], random search [10], reinforcement learning [11,12], Bayesian optimization [13–15], and evolutionary algorithms (EA) [16,17]. The first approach for evolutionary NAS comes from Miller et al. [18], who used the genetic algorithm (GA) method to evolve artificial neural network (ANN) architectures for specific tasks. Then, Stanley and Risto [19] introduced NeuroEvolution of Augmenting Topologies (NEAT), a GA-based method for evolving ANN architectures. Real et al. [20] found that evolution performs equally well as reinforcement learning in terms of accuracy but created better mid-term performance and more modest models. Young [21] proposed MENNDL, a framework for evolving a fixed three-layer CNN architecture using GA, but the search space was limited, consisting of only six hyperparameters. Recently, researchers have applied evolutionary NAS to CNN architecture design, achieving state-of-the-art performance. Loshchilov et al. [22] proposed the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) for CNN hyperparameters tuning using the MNIST dataset. However, their

work did not consider some of the convolutional parameters (e.g., filter sizes, activation functions, etc.) In [23], the authors proposed a differentiable version of the Compositional Pattern Producing Network, named the DPPN. The DPPNs are generated using microbial GAs and capable of replicating CNN architectures. Baldominos [24] presented a common framework for the automatic design of CNN topologies and developed two solutions based on GA and grammatical evolution. They used the MNIST dataset to evaluate their proposals and achieved highly competitive results based on the latest state of the art. Swarm algorithms have also been adopted to automatically design CNN architectures. Wang et al. [25] proposed a multi-objective particle swarm optimization (PSO) method to evolve CNN architectures for image classification. Byla et al. [26] proposed ant colony optimization (ACO) to collectively search for the best CNN architecture for image classification problems. Other extraordinary works that employed nature-inspired methods for evolving CNN architectures are detailed in the following [27–42].

Manually designing optimal CNN architectures is a challenging task due to the vast search space and the critical design decisions that affect the model's accuracy and speed. To overcome this challenge, we propose a novel neural architecture search (NAS) framework that utilizes the clonal selection algorithm (CSA), an immune-inspired optimization algorithm that has shown promise in other optimization tasks. CSA is a diversity-generating algorithm that aims to find solutions that not only perform well but also differ from existing solutions. This approach is particularly useful for exploring the vast search space of CNN architectures and identifying high-quality designs for image classification tasks. Our proposed method expands the search space to include other sets of layers and hyperparameters, such as pooling layers, number of epochs, batch size, optimizers, and activation functions, to efficiently explore the search space and discover optimal CNN architectures that generalize well across different datasets. We evaluate the proposed method on challenging and imbalanced datasets such as EMNIST, which contains both digits and letters, with a larger amount of data. In contrast to other works that often use standard datasets such as CIFAR-10 [43] and MNIST [44] to benchmark their proposals, our work tests the proposed method on more challenging datasets. Our proposed method achieves competitive performance on the EMNIST datasets and demonstrates the effectiveness of using CSA in NAS for CNN architecture design.

3. Background

3.1. CNN

A CNN is a type of deep feed-forward neural network that has proven to be effective for various applications, including image classification, object detection, segmentation, and many others [45]. One of the earliest and most influential CNN models was LeNet-5, developed by Yann LeCun [46]. It was originally designed for handwritten and machine-printed character recognition, but its success led to its use in many other areas, including facial recognition and self-driving cars. The architecture of a CNN is unique compared to other types of neural networks, such as multi-layer perceptrons. CNN layers are arranged in three dimensions, including width, height, and depth. Neurons in one layer only connect to some of the neurons in the following layer, rather than all of them [47]. A simple CNN architecture consists of several layers, with each layer transforming one activation volume into another through a differentiable function [48]. The four main types of layers used to build CNN architectures include the convolutional layer, the ReLU layer, the pooling layer, and fully connected. Figure 1 illustrates a general architecture of a CNN. It consists of multiple layers, each with a specific function in transforming the input image to a set of class probabilities. The input image is first processed by a set of convolutional layers, followed by ReLU layers for nonlinearity and pooling layers to downsample the feature maps. Finally, the fully-connected layers are used to obtain the class probabilities.

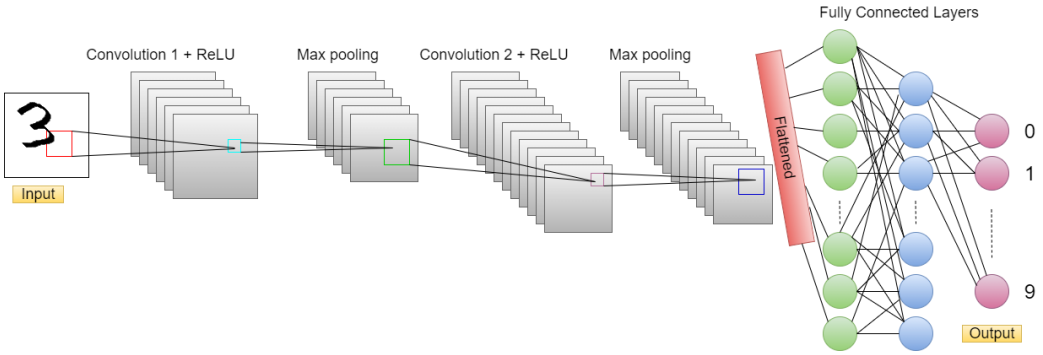

Figure 1. Illustration of CNN architecture.

3.1.1. Convolutional Layer

The convolutional layer is a fundamental building block of CNNs. It performs convolution, a mathematical operation that combines two functions to form a third function. In a CNN, convolution is performed on the input image with the application of a filter (or kernel) to produce a feature map as an output [45]. The filter is slid over the input image, and element-wise multiplication between the filter values and their corresponding values in the input image is performed at each location. These values are then summed and used as input to the feature map.

In a convolutional layer, there are usually multiple filters that convolve with the input image and generate different feature maps. These features are then assembled to form the final output of the convolutional layer. To perform the convolution operation, the number of filters and their spatial size must be specified [48]. Padding and stride also need to be determined. Padding is necessary to avoid the feature maps' spatial size from shrinking as the network moves to deeper layers. Zero-padding is often used to keep the feature map the same spatial size as the input image. The amount of zero-padding (P) that is needed to achieve this is given by Equation (1):

$$P = \frac{F-1}{2} \tag{1}$$

where F is the width or height of the filter. The stride parameter controls how the filter convolves around the input volume. A stride size of one means the filter slides one pixel at a time, while a stride size of two means the filter slides two pixels at a time. Higher stride values produce smaller feature maps. When applying a convolution with a kernel size of F on an input image with size W_1, zero-padding size P, and stride size S, the size of the output or feature map W_2 can be calculated as follows:

$$W_2 = \frac{W_1 - F + 2P}{S} + 1 \tag{2}$$

3.1.2. ReLU

In order to introduce nonlinearity into a CNN, each generated feature map in the convolutional layer is passed individually through a rectified linear unit (ReLU) function. The ReLU function is defined as:

$$f(z) = \max(0, z) \tag{3}$$

where z is the input to the ReLU function. ReLU is often preferred over other activation functions (such as sigmoid or tanh) because it can train networks faster without a substantial penalty for generalization accuracy [45].

3.1.3. Pooling Layer

The pooling layer is inserted between the convolutional layers in a CNN architecture. The main function of the pooling layer is to reduce the dimensionality of the rectified

feature maps to reduce the number of parameters and computation in the network, which in turn reduces the training time and controls overfitting [48]. Pooling is often performed by a simple operation such as average or max pooling. Max pooling has shown better performance and faster convergence compared to other types of pooling [49]. Therefore, recent work generally tends towards using max pooling in CNN architectures.

Max pooling is applied by dividing the input feature map into non-overlapping rectangular regions and taking the maximum value in each region as the output. The rectangular regions can be of various sizes, but the most common choice is to use 2×2 regions with a stride of 2. This downsamples the feature maps by a factor of two, which results in smaller feature maps and reduces the computational complexity of the network.

Average pooling computes the average value of each rectangular region instead of the maximum value. It can also downsample the feature maps but is less commonly used than max pooling in modern CNN architectures.

The pooling layer has no learnable parameters, and its main purpose is to introduce translation invariance and make the network more robust to small translations in the input image. By downsampling the feature maps, the pooling layer forces the network to learn more robust features that are less sensitive to the exact location of the object in the image.

3.1.4. Fully-Connected Layer

The fully-connected layer (FCL) [2] is typically located at the end of a CNN architecture. Its input is a flattened vector obtained from the output of the previous convolutional or pooling layer. The FCL's output is then passed through the last layer, which is usually a softmax layer. The softmax function is a common choice for the output layer of a classification network, as it produces a probability distribution over the possible classes [50]. The softmax function takes a vector of arbitrary real-valued scores and produces a vector of probabilities that sums to one. The formula for the softmax function is:

$$P(i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{4}$$

where $P(i)$ is the probability that the input image belongs to class i, z_i is the score for class i, and K is the total number of classes. The softmax function effectively converts the scores output by the FCL into a set of probabilities that can be interpreted as the confidence of the model in its classification decision.

3.2. CSA

The CSA is a computational optimization technique inspired by the principles of clonal selection and affinity maturation in the adaptive immune system [51]. The algorithm works by initializing a random population of antibodies to a given antigen or problem [6]. These antibodies represent potential solutions to the optimization problem. Each antibody is evaluated by its affinity score or fitness value, which reflects how well it performs in finding the optimal solution. Antibodies with the highest affinity scores are selected and cloned to generate a new set of antibodies. The number of clones produced for each antibody is proportional to its affinity score. This new set of antibodies is then subjected to an affinity maturation process, where a mutation operator is applied to improve the solutions. The mutation rate is usually inversely proportional to the affinity score. The best-performing antibody from the mutated set is then selected to replace the original antigen, and the process is repeated until a termination criterion is met [52].

The CSA algorithm has been shown to be robust, self-tuning, and highly effective in solving various optimization problems, especially those with multiple local optima [53]. The CSA algorithm has been applied in diverse areas, including data clustering, image segmentation, and feature selection, among others [54]. Although inspired by the biological immune system, CSA is a heuristic algorithm and does not model the immune system accurately. Nonetheless, the algorithm's effectiveness in solving optimization problems has led to its popularity in the field of optimization and machine learning.

4. Automated CNN Design Using CSA

The design of CNN networks can be a complex task, as it involves many hyperparameters that determine the effectiveness of the model in solving a specific problem. To simplify this process, a CSA-based approach is proposed in this section to automatically generate the complete design of the CNN and improve its performance in solving a specific problem, such as the classification of handwritten letters and digits. The hyperparameters related to the CNN architecture and their ranges considered in the proposed approach are shown in Table 1. Some hyperparameters, such as the padding and stride of convolutional layers, are fixed. The last layer of the CNN architecture will always be a softmax layer with the number of neurons corresponding to the dataset's number of classes or labels. For instance, for the EMNIST Digits dataset, there are 10 neurons in the output layer (softmax layer), which represents the 10 possible classes in the dataset. The output of each neuron corresponds to a probability value for each class.

Table 1. CNN architecture hyperparameters and range.

Hyperparameter	Range
# Epochs	[1–49]
Batch Size	[1–255]
# Conv layers	[1–9]
# Filters in each conv layer	[1–65]
Filters size in each conv layer	[1–9]
Activation function in each conv layer	{Relu(1), Tanh (2), Linear (3), Sigmoid (4)}
Max pooling size (if any) after each conv layer	{False (1), True (2)}
Pooling size of each max pooling layer	[2–9]
# Dense layers	[1–9]
# Neurons for each dense layer	[1–65]
Activation function in each dense layer	{Relu (1), Tanh (2), Linear (3), Sigmoid (4)}
Optimizer	{SGD (1), Adadelta (2), RMSprop (3), Adam (4), Nadam (5)}
Learning rate	[0.00001, 0.0001, 0.001, 0.01]

Once the key hyperparameters of the CNN design are determined, an efficient search procedure for these hyperparameters is performed using CSA. CSA is a robust optimization algorithm that can perform heuristic searches in large and complex spaces if the encoding of the search space is suitable. Therefore, an integer encoding scheme for the CSA method is proposed and implemented. This encoding scheme accurately and efficiently encodes all possible CNN architectures and hyperparameters, allowing for a variety of mutation applications. In addition to the encoding scheme, a CSA affinity assessment method must be defined and implemented. This method is used to measure the affinity between the antigen and the antibody during the affinity maturation process. The CSA algorithm then uses the affinity scores to select and clone the best antibodies for further mutation. The overall procedure is illustrated in Figure 2. In the upcoming subsections, we detail the steps involved in our proposed method, which include encoding the antigen, generating antibodies, evaluating affinity and selecting the best candidates, cloning and mutating the selected candidates, and evaluating the resulting clones.

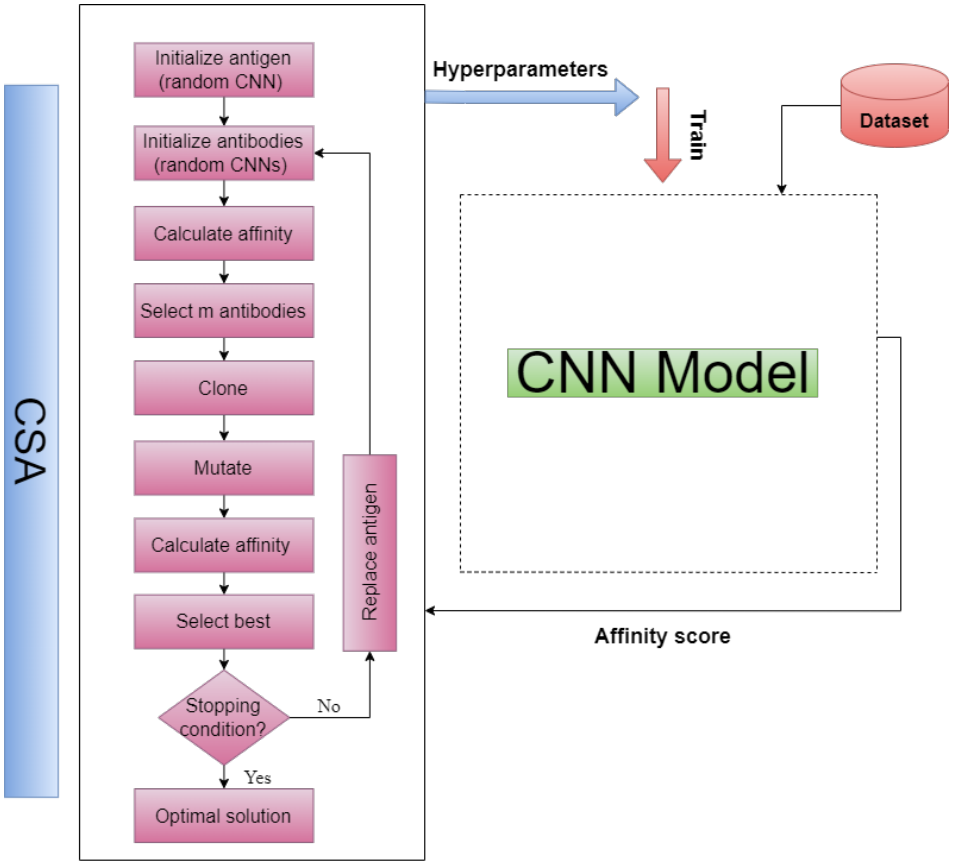

Figure 2. Proposed CSA–CNN architecture optimization procedure.

4.1. Antigen Encoding

In the proposed CSA–CNN architecture optimization procedure, the first step is to encode the CNN architecture as an antigen. The antigen is a set of hyperparameters that represent a potential solution to the optimization problem. The optimal values of these hyperparameters must be found using the CSA algorithm to create a CNN architecture with the highest test classification accuracy. The antigen is represented by a set of 13 hyperparameters, which are encoded as an integer vector. These hyperparameters form the genes of the antigen. Figure 3 shows an example of a randomly generated antigen, which represents a CNN architecture. The values of the hyperparameters in the antigen are restricted to the ranges defined in Table 1.

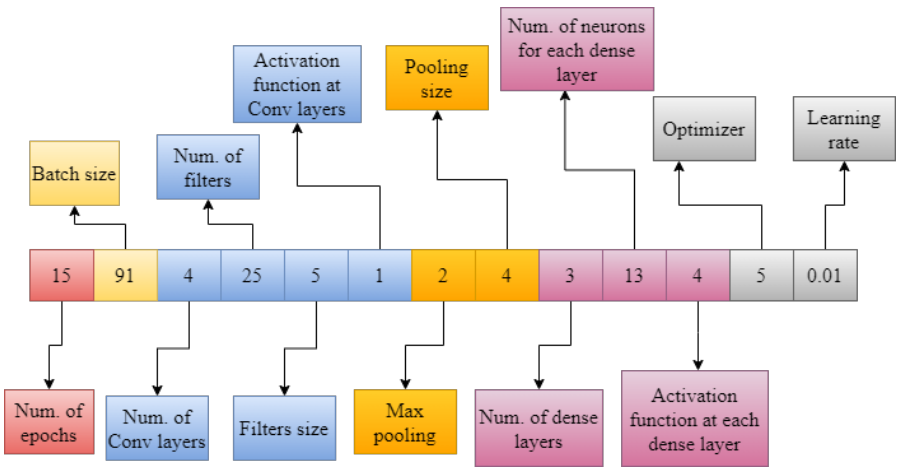

Figure 3. Randomly generated antigen, representing a CNN architecture.

4.2. Antibodies

The size of the antibody population is an important factor in the performance of the CSA–CNN optimization procedure. In this work, we set the population size to 20, meaning that 20 different candidate solutions (antibodies) are generated in each iteration of the algorithm. These antibodies are randomly created in response to the antigen initialized in the previous step, as depicted in Figure 4. Each antibody is also encoded as an integer vector, similar to the antigen, and represents a unique CNN architecture.

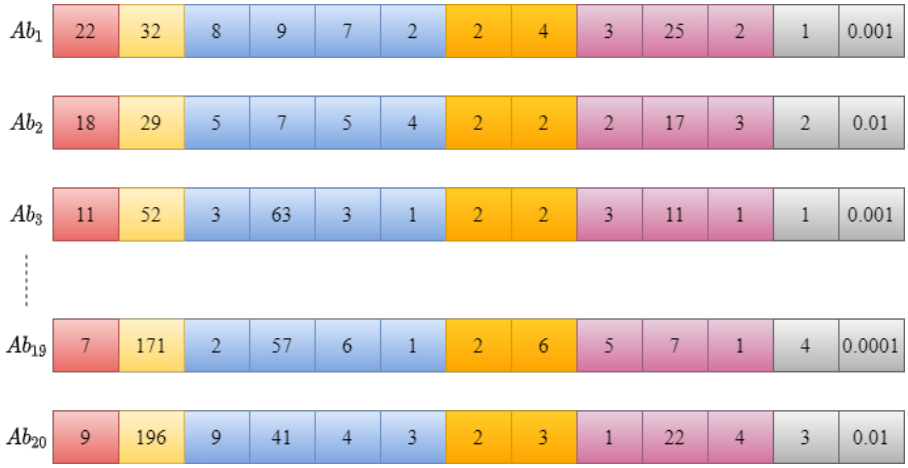

Figure 4. A randomly generated initial population of antibodies (CNN architectures).

4.3. Affinity Evaluation and Selection

After the creation of the population of antibodies, the affinity of each antibody in the population is evaluated by measuring the accuracy of the CNN model on a held-out test dataset. The weights and biases of the CNN model are trained using a specific set of hyperparameters determined by the antibody genes in the training dataset, and the accuracy of the test classification is computed as the ratio of correctly classified test instances to the total number of test instances. The higher the accuracy is, the higher the affinity of the antibody will be. The primary antigen, which is the initial solution, is also evaluated using the same method. Antibodies with higher affinity than the antigen are selected, while those with lower affinity are removed from the population, reducing the execution time of the algorithm. The selected antibodies proceed to the next steps of the procedure, cloning and mutation, to produce new candidate solutions.

4.4. Cloning and Mutation

The next step in the proposed CSA–CNN architecture optimization procedure is cloning and mutation. After selecting the antibodies with higher affinity than the antigen, a set number of clones for each selected antibody is created. The number of clones is determined using the following formula:

$$N_{clones} = \beta \times N_{antibodies}, \quad (5)$$

where, $N_{antibodies}$ is the population size of antibodies, and β is the clone factor, which is a hyperparameter that determines the number of clones. In this work, we use a clone factor of 0.3. A higher affinity value results in more clones being created.

After creating the clones, a mutation operation is performed on each clone. The mutation operation involves randomly selecting genes from the clone and replacing their values with new ones obtained using truncated Gaussian distribution (TGD). TGD is similar to normal distribution but is truncated at a defined range. The range for TGD is determined based on the ranges of the hyperparameters in the antigen. The mutation process is controlled by a mutation rate, which is inversely proportional to the affinity of the antibody. The mutation rate determines the probability of individual component

mutation for a given candidate solution, and it is calculated as the exponential of negative product of the mutation rate and the candidate's affinity value, that is, $\exp(-\rho \cdot f)$. Here, f represents the candidate's normalized fitness value, and ρ is the user-defined mutation rate hyperparameter. The lower the candidate's affinity value is, the higher the probability of mutation will be. An illustration of TGD mutation for the antibody Ab_{20} is shown in Figure 5.

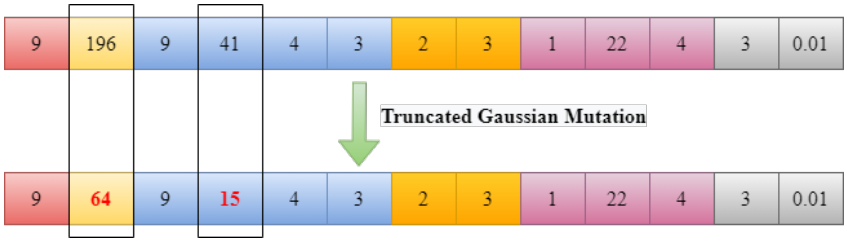

Figure 5. Truncated Gaussian mutation applied to CNN.

After the mutation operation, the fitness of each clone is evaluated using the same affinity evaluation method used in the previous step. The clone with the highest affinity is then selected to replace the original antigen, and the process repeats with the new antigen until a termination condition is met, such as finding the optimal CNN architecture or reaching a maximum number of iterations (generations).

5. Experimental Design

5.1. EMNIST Dataset

The performance of the proposed method in discovering the optimal architecture of a CNN is evaluated using the EMNIST dataset. While the MNIST dataset is commonly used to evaluate the performance of deep learning algorithms, achieving high accuracy with CNN models on this dataset is relatively easy [55–59]. To provide a more challenging and complex dataset for new benchmarks in deep learning, Cohen et al. [60] developed the EMNIST (Extended MNIST) dataset in 2017. The EMNIST dataset comprises lowercase letters (a-z), uppercase letters (A-Z), and handwritten digits (0–9) and is derived from the NIST Special Database 19 [61]. The dataset was transformed into 28×28 pixel binary images with 8-bit grayscale resolution that match the structure of the MNIST dataset. The conversion process involves applying a Gaussian blur filter, extracting the area around the digit, centering the digit in a square box while maintaining the aspect ratio, adding padding, and finally downsampling the image to 28×28 pixels using bi-cubic interpolation.

The EMNIST dataset includes two labeling schemata, which have been ported from the NIST SD 19 to the EMNIST dataset: the By_Class and By_Merge datasets, which, respectively, have 62 and 47 classes. Additionally, EMNIST generated four other datasets: the EMNIST Balanced dataset, which balances the number of instances per class, the EMNIST Letters dataset, which contains only letters, the EMNIST Digits dataset, and the EMNIST MNIST dataset, which contains only digits and has the same classes as the original MNIST dataset. Table 2 presents the specifications of each EMNIST dataset, including the total number of instances/images, the training/testing splits, and the number of classes.

Table 2. Specification of the EMNIST classification datasets.

Dataset	# Images	# Training Images	# Test Images	# Classes
By_Class	814,255	697,932	116,323	62 (unbalanced)
By_Merge	814,255	697,932	116,323	47 (unbalanced)
Balanced	131,600	112,800	18,800	47 (balanced)
Letters	145,600	124,800	20,800	26 (balanced)
Digits	280,000	240,000	40,000	10 (balanced)
MNIST	70,000	60,000	10,000	10 (balanced)

5.2. Experimental Setup

The proposed approach was designed, trained, and validated in Python 3.7 using Keras running on top of TensorFlow. We used a Dell desktop workstation with an Intel (R) Core (TM) i77800X CPU @ 3.50GHz, 32GB DDR4 RAM, and an 8GB Nvidia Quadro P4000 GPU for experimentation. The CSA algorithm was used to generate a successful CNN architecture, which took approximately 3.63 h. We selected EMNIST Digits, which contains 240,000 images as a training set and 40,000 images as a test set, to evaluate the ability of the proposed method to discover the optimal CNN architecture. Once the architecture was found, we reused it for the other five EMNIST datasets, except for the last layer (softmax layer), which was modified to have the same number of neurons as the number of classes in the dataset. The architecture was then trained from scratch to learn new weights and biases for each individual dataset. Our hypothesis was that the obtained architecture from EMNIST Digits would provide accurate classifications since the data structure in the EMNIST Digits dataset is equivalent, and the domain is quite similar to the other EMNIST datasets. The categorical cross-entropy was adopted as the loss function. This function is commonly used in multiclass classification problems.

6. Results

The proposed CSA algorithm successfully discovered an optimal CNN architecture for the EMNIST Digits dataset, achieving a classification accuracy of 99.74% on the test set. Table 3 presents the hyperparameters and their corresponding optimal values of the discovered architecture. We further evaluated the robustness of the optimized architecture by testing it on the other splits of the EMNIST dataset. As each dataset has a different number of classes, we adjusted the softmax layer to have the same number of neurons as the number of classes in each dataset. Figures 6 and 7 provide a plot of accuracy and loss, respectively, on the training and test sets for each of the six EMNIST datasets. The figures demonstrate the performance of the proposed CNN models on the six datasets and provide insights into how the models generalize on unseen data. We implemented the early stopping method to stop the training of the CNN models as soon as their performance stopped improving on a specific dataset in the validation set. This strategy saved time, effort, and computational resources, especially when some datasets (as observed for EMNIST MNIST and EMNIST By_Class) achieved their best performance in less than six epochs. The results show that the proposed CNN models achieved high accuracy and low loss on most of the datasets, and the early stopping method was effective in preventing overfitting. These results demonstrate the potential of the proposed CSA algorithm in discovering optimal CNN architectures that can generalize well on a variety of datasets.

Table 4 provides a comprehensive summary of the performance of each dataset's tuned architecture in terms of train/test accuracy and train/test loss, as well as the total number of trainable parameters in the model. The table reveals that the CNN architecture of the By_Class dataset has more parameters than the other architectures, primarily because it contains 64 neurons in the last layer, which corresponds to the number of classes in the By_Class dataset. It is worth noting that the tuned architecture's performance varies across the different datasets. The EMNIST Digits dataset has the highest accuracy for both the train and test sets, with a value of 99.82% and 99.74%, respectively, while the EMNIST Balanced dataset has the lowest accuracy with a train and test accuracy of 89.95% and 88.12%, respectively. In terms of the number of trainable parameters, the architectures for all the datasets have a similar number of parameters, ranging from 241,299 to 242,339.

Table 3. Optimal hyperparameters of the CNN architecture discovered by the proposed CSA method.

Hyperparameter	Optimal Value
# Epochs	6
Batch Size	151
# Conv layers	4
# Filters in each conv layer	56
Filters size in each conv layer	5×5
Activation function in each conv layer	Relu
Max pooling size (if any) after each conv layer	True
Pooling size of each max pooling layer	2×2
# Dense layers	1
# Neurons for each dense layer	19
Activation function in each dense layer	Relu
Optimizer	Adam
Learning rate	0.001

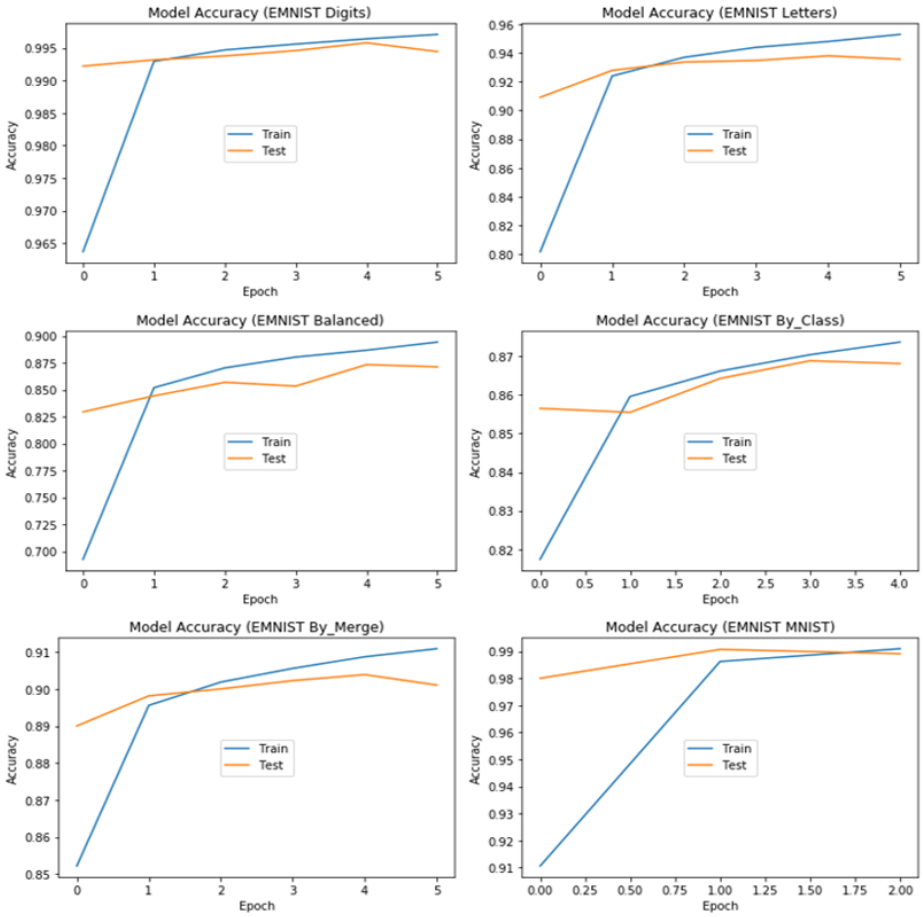

Figure 6. Training and test accuracy of the tuned architecture for each dataset.

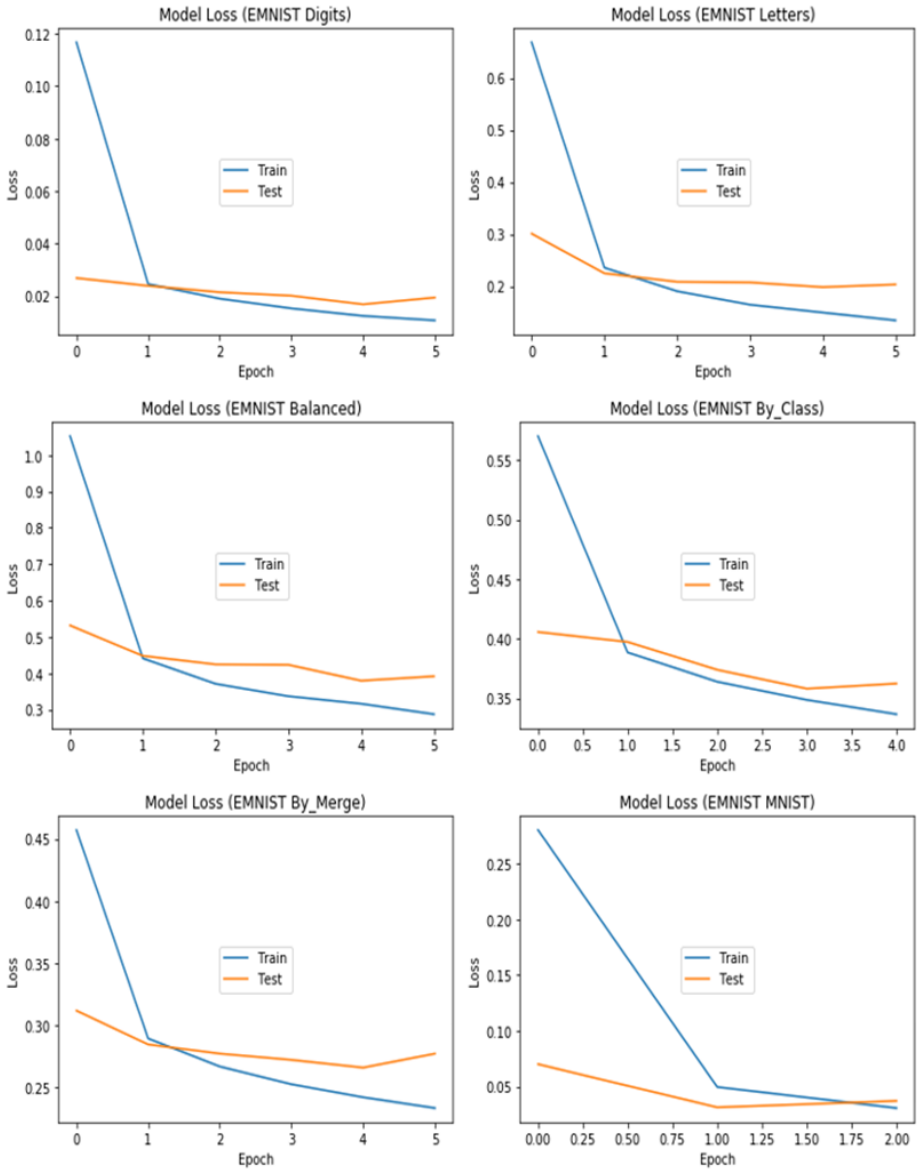

Figure 7. Training and test loss of the tuned architecture for each dataset.

Table 4. Performance summary of each dataset.

Dataset	Train Accuracy	Test Accuracy	Train Loss	Test Loss	# Parameters
Digits	99.82%	99.74%	0.0107	0.0194	241,299
Letters	95.46%	93.95%	0.1344	0.2034	241,619
Balanced	89.95%	88.12%	0.2872	0.3915	242,039
By_Class	87.80%	86.82%	0.3366	0.3624	242,339
By_Merge	91.29%	90.11%	0.2332	0.2769	242,039
MNIST	99.11%	98.91%	0.0305	0.0369	241,299

6.1. Benchmarking with Existing Methods

In Table 5, we compare the state-of-the-art accuracy of our proposed approach (CSA–CNN) with other methods that have been evaluated on EMNIST datasets. For the Digits dataset, the best-reported accuracy in the existing literature is 99.62%, achieved using manually designed parallelized CNN [62]. In contrast, our model achieved an accuracy of 99.74%. Our CSA–CNN outperformed all other methods on the By_Merge, Balanced, and Letters datasets, achieving accuracy of 90.11%, 88.12%, and 93.95%, respectively. Given

that we used the architecture discovered by CSA for the Digits dataset, it is possible to obtain even better results on the By_Class dataset by searching for an optimal architecture. Nonetheless, our proposed method still achieved a competitive performance of 86.82% on the By_Class dataset, with only a 1% decrease compared to manually designed CNN models [63]. An important advantage of the proposed method is that that our model achieves high accuracy with a minimal number of network parameters indicating that it is more efficient and resource-friendly compared to other methods that require a larger number of parameters to achieve a similar performance. This is particularly important for real-world applications where computational resources and energy consumption are important considerations. In addition, having a smaller model size can also improve the model's generalization performance, which is the ability to perform well on unseen data. Therefore, the minimal number of network parameters is an important factor to consider when evaluating the performance of deep learning models.

Table 5. Comparative accuracy of EMNIST classification methods.

Method	By_Class	By_Merge	Balanced	Letters	Digits
Our CSA–CNN	86.82%	90.11%	88.12%	93.95%	99.74%
OPIUM [60]	69.71%	72.57%	78.02%	85.15%	95.90%
Linear classifier [60]	51.80%	50.51%	50.93%	55.78%	84.70%
OptConv+Log+Perc [64]	–	–	87.69%	93.65%	99.43%
OptConv+Perc [64]	–	–	84.68%	91.92%	98.29%
DWT-DCT-SVM [65]	–	–	–	89.51%	97.74%
Autoencoder [66]	–	–	–	91.27%	–
HM2-BP [67]	–	–	85.57%		
CNN [63]	87.10%	–	–	–	–
Parallelized CNN [62]	–	–	–	–	99.62%
CNN [68]	–	–	87.18%	93.63%	99.46%
EDEN [69]	–	–	–	88.30%	99.30%

6.2. Discussion

The results presented in this paper demonstrate the effectiveness of the proposed NAS framework that uses the CSA to automatically design CNN architectures for image classification problems. Our method outperforms many existing approaches and produces high-quality CNN architectures that achieve state-of-the-art performance on six benchmark datasets. One of the advantages of the proposed NAS framework is that it provides a systematic and automated approach to designing CNN architectures, which can reduce the amount of manual effort and expert knowledge required for model design. Moreover, the use of integer vector representation and truncated Gaussian mutation in the CGA algorithm makes the search process more efficient and effective, as it allows us to explore the search space more thoroughly and avoid local optima.

The results also highlight the importance of early stopping to prevent overfitting, as we were able to achieve high accuracy with a low number of epochs on some of the datasets. This suggests that the proposed method is able to discover optimal architectures quickly, without the need for extensive training. However, we also observed some limitations of the proposed NAS framework. For example, the method may generate architectures that are difficult to interpret or transfer to other applications. Additionally, the proposed method may not be suitable for problems with limited computational resources or small datasets, as the search process can be computationally expensive and requires large amounts of training data.

Therefore, developing more efficient and scalable NAS frameworks may be necessary to address these limitations. Overall, our findings suggest that the proposed NAS framework is a promising approach to automatically designing high-quality CNN architectures for image classification problems. However, more work is needed to address the limitations of the proposed method and explore its full potential. By improving the efficiency and effectiveness of NAS frameworks, we can accelerate the development of deep learning

models and facilitate the discovery of novel architectures that can solve complex and important problems in various domains.

7. Conclusions

In this study, we have presented a novel NAS-based CSA framework to automatically design CNN architectures for image classification tasks. Our proposed method achieved state-of-the-art accuracy on six different EMNIST benchmark datasets for handwriting recognition, outperforming most of the existing approaches. The superiority of our model comes not only from its high classification accuracy but also from the fact that it has a minimal number of network parameters, indicating that it is more efficient and resource-friendly compared to other methods that require a larger number of parameters to achieve similar performance. Our work has demonstrated the potential of an NAS-based CSA for automating the process of deep learning model design, which can replace the challenging and time-consuming process of manual feature engineering. This provides a low-cost, simple, and reusable architecture that can work with multiple datasets while also providing cutting-edge and highly competitive performance. In future work, we plan to extend the proposed NAS-based CSA framework to other computer vision tasks, such as object detection and semantic segmentation, and evaluate its effectiveness on larger and more complex datasets. We also aim to evaluate the use of more sophisticated affinity evaluation methods, such as reinforcement learning or surrogate models, to more accurately and efficiently evaluate the fitness of each developed architecture. These methods will help to improve the quality and diversity of the generated architectures and enhance the effectiveness of the proposed NAS framework.

Author Contributions: Conceptualization, A.A.B.; formal analysis, A.A.B.; investigation, A.A.B., D.K., M.A.-k. and E.A.-s.; methodology, A.A.B.; resources, A.A.B., D.K., M.A.-k. and E.A.-s.; validation, A.A.B., D.K., M.A.-k., and E.A.-s.; visualization, A.A.B.; writing—review and editing, A.A.B., D.K., M.A.-k., and E.A.-s. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: https://www.nist.gov/itl/products-and-services/emnist-dataset, accessed on 15 January 2023.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef]
2. Al Bataineh, A. A comparative analysis of nonlinear machine learning algorithms for breast cancer detection. *Int. J. Mach. Learn. Comput.* **2019**, *9*, 248–254. [CrossRef]
3. Goodfelow, I.; Bengio, Y.; Courville, A. *Deep Learning (Adaptive Computation and Machine Learning Series)*; MIT Press: London, UK, 2016.
4. Al Bataineh, A.; Mairaj, A.; Kaur, D. Autoencoder based semi-supervised anomaly detection in turbofan engines. *Int. J. Adv. Comput. Sci. Appl. (IJACSA)* **2020**, *11*. [CrossRef]
5. Al Bataineh, A.; Jarrah, A. High Performance Implementation of Neural Networks Learning Using Swarm Optimization Algorithms for EEG Classification Based on Brain Wave Data. *Int. J. Appl. Metaheuristic Comput. (IJAMC)* **2022**, *13*, 1–17. [CrossRef]
6. de Castro, L.N.; Von Zuben, F.J. Learning and optimization using the clonal selection principle. *IEEE Trans. Evol. Comput.* **2002**, *6*, 239–251. [CrossRef]
7. Castro, L.N.; De Castro, L.N.; Timmis, J. *Artificial Immune Systems: A New Computational Intelligence Approach*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2002.
8. Elsken, T.; Metzen, J.H.; Hutter, F. Neural architecture search: A survey. *J. Mach. Learn. Res.* **2019**, *20*, 1–21.
9. Hershey, S.; Chaudhuri, S.; Ellis, D.P.; Gemmeke, J.F.; Jansen, A.; Moore, R.C.; Plakal, M.; Platt, D.; Saurous, R.A.; Seybold, B.; et al. CNN architectures for large-scale audio classification. In Proceedings of the 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), New Orleans, LA, USA, 5–9 March 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 131–135.

10. Liashchynskyi, P.; Liashchynskyi, P. Grid search, random search, genetic algorithm: A big comparison for nas. *arXiv* **2019**, arXiv:1912.06059.
11. Zoph, B.; Le, Q.V. Neural architecture search with reinforcement learning. *arXiv* **2016**, arXiv:1611.01578.
12. Baker, B.; Gupta, O.; Naik, N.; Raskar, R. Designing neural network architectures using reinforcement learning. *arXiv* **2016**, arXiv:1611.02167.
13. Pelikan, M.; Goldberg, D.E.; Cantú-Paz, E. BOA: The Bayesian optimization algorithm. In Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99, Orlando, FL, USA, 13–17 July 1999; Citeseer: Princeton, NJ, USA, 1999; Volume 1, pp. 525–532.
14. Snoek, J.; Larochelle, H.; Adams, R.P. Practical bayesian optimization of machine learning algorithms. *arXiv* **2012**, arXiv:1206.2944.
15. Lopes, V.; Alexandre, L.A. HMCNAS: Neural Architecture Search using Hidden Markov Chains and Bayesian Optimization. *arXiv* **2020**, arXiv:2007.16149.
16. Yao, X.; Liu, Y. A new evolutionary system for evolving artificial neural networks. *IEEE Trans. Neural Netw.* **1997**, *8*, 694–713. [CrossRef] [PubMed]
17. Kassahun, Y.; Sommer, G. Efficient reinforcement learning through Evolutionary Acquisition of Neural Topologies. In Proceedings of the ESANN, Bruges, Belgium, 27–29 April 2005; pp. 259–266.
18. Miller, G.F.; Todd, P.M.; Hegde, S.U. Designing Neural Networks using Genetic Algorithms. In Proceedings of the ICGA, Fairfax, VA, USA, 15 June 1989; Volume 89, pp. 379–384.
19. Stanley, K.O.; Miikkulainen, R. Evolving neural networks through augmenting topologies. *Evol. Comput.* **2002**, *10*, 99–127. [CrossRef] [PubMed]
20. Real, E.; Aggarwal, A.; Huang, Y.; Le, Q.V. Regularized evolution for image classifier architecture search. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; Volume 33, pp. 4780–4789.
21. Young, S.R.; Rose, D.C.; Karnowski, T.P.; Lim, S.H.; Patton, R.M. Optimizing deep learning hyper-parameters through an evolutionary algorithm. In Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments, Austin, TX, USA, 15–20 November 2015; pp. 1–5.
22. Loshchilov, I.; Hutter, F. CMA-ES for hyperparameter optimization of deep neural networks. *arXiv* **2016**, arXiv:1604.07269.
23. Fernando, C.; Banarse, D.; Reynolds, M.; Besse, F.; Pfau, D.; Jaderberg, M.; Lanctot, M.; Wierstra, D. Convolution by evolution: Differentiable pattern producing networks. In Proceedings of the Genetic and Evolutionary Computation Conference 2016, Denver, CO, USA, 20–24 July 2016; pp. 109–116.
24. Baldominos Gómez, A. Evolutionary design of deep neural networks. In Proceedings of the 2019 21st International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), Timisoara, Romania, 4–7 September 2018.
25. Wang, B.; Sun, Y.; Xue, B.; Zhang, M. Evolving deep neural networks by multi-objective particle swarm optimization for image classification. In Proceedings of the Genetic and Evolutionary Computation Conference, Prague, Czech Republic, 13–17 July 2019; pp. 490–498.
26. Byla, E.; Pang, W. Deepswarm: Optimising convolutional neural networks using swarm intelligence. In Proceedings of the UK Workshop on Computational Intelligence, Portsmouth, UK, 4–6 September 2019; Springer: Berlin/Heidelberg, Germany, 2019; pp. 119–130.
27. Stanley, K.O.; D'Ambrosio, D.B.; Gauci, J. A hypercube-based encoding for evolving large-scale neural networks. *Artif. Life* **2009**, *15*, 185–212. [CrossRef]
28. Verbancsics, P.; Harguess, J. Image classification using generative neuro evolution for deep learning. In Proceedings of the 2015 IEEE Winter Conference on Applications of Computer Vision, Waikoloa, HI, USA, 3–8 January 2015; IEEE: New York, NY, USA, 2015; pp. 488–493.
29. Baldominos, A.; Saez, Y.; Isasi, P. Evolutionary design of convolutional neural networks for human activity recognition in sensor-rich environments. *Sensors* **2018**, *18*, 1288. [CrossRef]
30. Suganuma, M.; Shirakawa, S.; Nagao, T. A genetic programming approach to designing convolutional neural network architectures. In Proceedings of the Genetic and Evolutionary Computation Conference, Berlin, Germany, 15–19 July 2017; pp. 497–504.
31. Xie, L.; Yuille, A. Genetic cnn. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 1379–1388.
32. Desell, T. Large scale evolution of convolutional neural networks using volunteer computing. In Proceedings of the Genetic and Evolutionary Computation Conference Companion, Berlin, Germany, 15–19 July 2017; pp. 127–128.
33. Baldominos, A.; Saez, Y.; Isasi, P. Model selection in committees of evolved convolutional neural networks using genetic algorithms. In Proceedings of the International Conference on Intelligent Data Engineering and Automated Learning, Madrid, Spain, 21–23 November 2018; Springer: Berlin/Heidelberg, Germany, 2018; pp. 364–373.
34. Bataineh, A.S.A. A gradient boosting regression based approach for energy consumption prediction in buildings. *Adv. Energy Res.* **2019**, *6*, 91–101.
35. Wang, B.; Sun, Y.; Xue, B.; Zhang, M. A hybrid differential evolution approach to designing deep convolutional neural networks for image classification. In Proceedings of the Australasian Joint Conference on Artificial Intelligence, Wellington, New Zealand, 11–14 December 2018; Springer: Berlin/Heidelberg, Germany, 2018; pp. 237–250.

36. Wang, B.; Sun, Y.; Xue, B.; Zhang, M. Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification. In Proceedings of the 2018 IEEE Congress on Evolutionary Computation (CEC), Rio de Janeiro, Brazil, 8–13 July 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 1–8.
37. Fielding, B.; Zhang, L. Evolving image classification architectures with enhanced particle swarm optimisation. *IEEE Access* **2018**, *6*, 68560–68575. [CrossRef]
38. Bhandare, A.; Kaur, D. Designing convolutional neural network architecture using genetic algorithms. In Proceedings of the Proceedings on the International Conference on Artificial Intelligence (ICAI). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), Jinan, China, 9–10 August 2018; pp. 150–156.
39. Miikkulainen, R.; Liang, J.; Meyerson, E.; Rawal, A.; Fink, D.; Francon, O.; Raju, B.; Shahrzad, H.; Navruzyan, A.; Duffy, N.; et al. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*; Elsevier: Amsterdam, The Netherlands, 2019; pp. 293–312.
40. Al Bataineh, A.; Manacek, S. MLP-PSO hybrid algorithm for heart disease prediction. *J. Pers. Med.* **2022**, *12*, 1208. [CrossRef]
41. Sun, Y.; Xue, B.; Zhang, M.; Yen, G.G. Evolving deep convolutional neural networks for image classification. *IEEE Trans. Evol. Comput.* **2019**, *24*, 394–407. [CrossRef]
42. Sun, Y.; Xue, B.; Zhang, M.; Yen, G.G.; Lv, J. Automatically designing CNN architectures using the genetic algorithm for image classification. *IEEE Trans. Cybern.* **2020**, *50*, 3840–3854. [CrossRef] [PubMed]
43. Krizhevsky, A.; Hinton, G. Learning Multiple Layers of Features from Tiny Images. 2009. Available online: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf (accessed on 15 January 2023).
44. LeCun, Y. The MNIST Database of Handwritten Digits. 1998. Available online: http://yann.lecun.com/exdb/mnist/ (accessed on 15 January 2023).
45. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. *Commun. ACM* **2017**, *60*, 84–90. [CrossRef]
46. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
47. Al Bataineh, A.; Kaur, D. Optimal Convolutional Neural Network Architecture Design Using Clonal Selection Algorithm. *Int. J. Mach. Learn. Comput.* **2019**, *9*, 788–794. [CrossRef]
48. Karpathy, A. Cs231n convolutional neural networks for visual recognition. *Neural Netw.* **2016**, *1*.
49. Scherer, D.; Müller, A.; Behnke, S. Evaluation of pooling operations in convolutional architectures for object recognition. In Proceedings of the International Conference on Artificial Neural Networks, Thessaloniki, Greece, 15–18 September 2010; Springer: Berlin/Heidelberg, Germany, 2010; pp. 92–101.
50. Al Bataineh, A.; Kaur, D.; Jalali, S.M.J. Multi-Layer Perceptron Training Optimization Using Nature Inspired Computing. *IEEE Access* **2022**, *10*, 36963–36977. [CrossRef]
51. Brownlee, J. *Clonal Selection Theory & Clonalg-the Clonal Selection Classification Algorithm (CSCA)*; Swinburne University of Technology: Melbourne, Australia, 2005; p. 38.
52. Al Bataineh, A.; Kaur, D. Immunocomputing-Based Approach for Optimizing the Topologies of LSTM Networks. *IEEE Access* **2021**, *9*, 78993–79004. [CrossRef]
53. Bataineh, A.A.; Kaur, D. Immuno-Computing-based Neural Learning for Data Classification. *Int. J. Adv. Comput. Sci. Appl.* **2019**, *10*. [CrossRef]
54. Brownlee, J. *Clever Algorithms: Nature-Inspired Programming Recipes*; Jason Brownlee: Melbourne, Australia, 2011.
55. Kowsari, K.; Heidarysafa, M.; Brown, D.E.; Meimandi, K.J.; Barnes, L.E. Rmdl: Random multimodel deep learning for classification. In Proceedings of the 2nd International Conference on Information System and Data Mining, Lakeland, FL, USA, 9–11 April 2018; pp. 19–28.
56. Byerly, A.; Kalganova, T.; Dear, I. A branching and merging convolutional network with homogeneous filter capsules. *arXiv* **2020**, arXiv:2001.09136.
57. Assiri, Y. Stochastic optimization of plain convolutional neural networks with simple methods. *arXiv* **2020**, arXiv:2001.08856.
58. Hirata, D.; Takahashi, N. Ensemble learning in CNN augmented with fully connected subnetworks. *arXiv* **2020**, arXiv:2003.08562.
59. Mazzia, V.; Salvetti, F.; Chiaberge, M. Efficient-CapsNet: Capsule Network with Self-Attention Routing. *arXiv* **2021**, arXiv:2101.12491.
60. Cohen, G.; Afshar, S.; Tapson, J.; Van Schaik, A. EMNIST: Extending MNIST to handwritten letters. In Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN), Anchorage, AK, USA, 14–19 May 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 2921–2926.
61. Grother, P.J. NIST special database 19. *Handprinted Forms and Characters Database*; National Institute of Standards and Technology: Gaithersburg, MD, USA, 1995; p. 10.
62. Singh, S.; Paul, A.; Arun, M. Parallelization of digit recognition system using deep convolutional neural network on CUDA. In Proceedings of the 2017 Third International Conference on Sensing, Signal Processing and Security (ICSSS), Chennai, India, 4–5 May 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 379–383.
63. Mor, S.S.; Solanki, S.; Gupta, S.; Dhingra, S.; Jain, M.; Saxena, R. Handwritten text recognition: With deep learning and Android. *Int. J. Eng. Adv. Technol.* **2019**, *8*, 172–178.

64. Pad, P.; Narduzzi, S.; Kundig, C.; Turetken, E.; Bigdeli, S.A.; Dunbar, L.A. Efficient Neural Vision Systems Based on Convolutional Image Acquisition. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 13–19 June 2020; pp. 12285–12294.
65. Ghadekar, P.; Ingole, S.; Sonone, D. Handwritten digit and letter recognition using hybrid DWT-DCT with KNN and SVM classifier. In Proceedings of the 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), Pune, India, 16–18 August 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 1–6.
66. Wiyatno, R.; Orchard, J. Style memory: Making a classifier network generative. In Proceedings of the 2018 IEEE 17th International Conference on Cognitive Informatics & Cognitive Computing (ICCI* CC), Berkeley, CA, USA, 16–18 July 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 16–21.
67. Jin, Y.; Zhang, W.; Li, P. Hybrid macro/micro level backpropagation for training deep spiking neural networks. *arXiv* **2018**, arXiv:1805.07866.
68. Cavalin, P.; Oliveira, L. Confusion matrix-based building of hierarchical classification. In Proceedings of the Iberoamerican Congress on Pattern Recognition, Madrid, Spain, 19–22 November 2018; Springer: Berlin/Heidelberg, Germany, 2018; pp. 271–278.
69. Dufourq, E.; Bassett, B.A. Eden: Evolutionary deep networks for efficient machine learning. In Proceedings of the 2017 Pattern Recognition Association of South Africa and Robotics and Mechatronics (PRASA-RobMech), Bloemfontein, South Africa, 29 November–1 December 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 110–115.