|  |  |
| --- | --- |
| nature inspired cnn optimization | gROUP members  15140094 Mohsin Ashraf  14140062 Mushood Hanif  15140107 Mehvish Hameed  14140044 Rida Yaqoob  Supervisor  Syed Qamar Askari |

1. Introduction

A **C**onvolutional **N**eural **N**etwork is a deep neural network whose architecture most commonly contains several convolutions, pooling and fully connected layers. Several recent studies focus on developing a novel CNN architecture that achieves higher classification accuracy e.g. GoogleNet, ResNet and DesNet. Despite their success, designing CNN architectures is still a difficult task because many design parameters exist such as the depth of a network, the type and parameters of each layer and their connectivity.

CNN architectures have become more complex suggestive of the fact that a significant number of design parameters have to be tuned to realize best performance for the given dataset. In light of this situation, automatic design methods for CNNs are highly beneficial.

Algorithms inspired by evolution and other natural phenomenon have been traditionally applied to designing CNN architectures. Evolution based algorithms basically mimic Darwinian evolution by implementing concepts like “survival-of-the-fittest”, crossover and mutation over a significant number of generations. Metaheuristics are algorithms that mimic basic and instinctual actions performed by various living entities for survival e.g. PSO (Particle-swarm optimization). These algorithms have the ability to not depend on a known goal state for progression and also due to their dynamic nature, encoding CNN architectures into their population elements is relatively easy.

CNNs have become an area of highly concentrated research in terms of finding an optimal architecture for a given dataset and much progress has been made in terms of this field. In this paper, we propose a novel method to optimize CNNs using nature-inspired metaheuristics as specified in section 4. The next section defines related work previously done towards this research field that we’ve referred to. Section 5 will shed light on our testing methods and results and section 6 will be the conclusion and description of the future work we plan to do.

**IMAGENET**

Deep convolutional neural networks were trained to classify 1.2 million images in the ImageNet LSVRC-2010 contest into 1000 different classes. The CNN, which consisted of 60 mill. Parameters and 650,000 neurons and 5 convolutional layers and a final 1000-way softmax layer performed exceptionally and gave results ranked top-1 and top-5 in error rates.

To improve their performance, they have collected larger datasets, used more powerful models, and used better techniques for preventing overﬁtting. For example, the best error rate at the time on the MNIST digit-recognition task (<0.3%) approached human performance. But objects in realistic settings exhibit considerable variability, so to learn to recognize them it is necessary to use much larger training sets.

They trained one of the largest convolutional neural networks to date on the subsets of ImageNet used in the ILSVRC-2010 and ILSVRC-2012 competitions and achieved by far the best results ever reported on these datasets.

The images were collected from the web and labeled by human labelers using Amazon’s Mechanical Turk crowd-sourcing tool. Starting in 2010, as part of the Pascal Visual Object Challenge, an annual competition called the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) has been held. ILSVRC-2010 is the only version of ILSVRC for which the test set labels are available, so this is the version on which we performed most of our experiments. ImageNet consists of variable-resolution images, while our system requires a constant input dimensionality. Therefore, down-sampled the images to a ﬁxed resolution of 256 × 256.

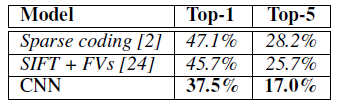
Deep convolutional neural networks with ReLUs train several times faster than their equivalents with tanh units. This is demonstrated in, which shows the number of iterations required to reach 25% training error on the CIFAR-10 dataset for a particular four-layer convolutional network

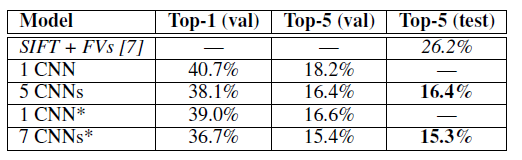
If at least some training examples produce a positive in putto a ReLU, learning will happen in that neuron. However, they still ﬁnd that the following local normalization scheme aids generalization.

Now they are ready to describe the overall architecture of our CNN. As depicted in Figure 2, the net contains eight layers with weights; the ﬁrst ﬁve are convolutional and the remaining three are fully connected. Our network maximizes the multinomial logistic regression objective, which is equivalent to maximizing the average a cross training cases of the log-probability of the correct label under the prediction distribution. The easiest and most common method to reduce overﬁtting on image data is to artiﬁcially enlarge the dataset using label-preserving transformations. They employ two distinct forms of data augmentation. The ﬁrst form of data augmentation consists of generating image translations and horizontal reﬂections. We do this by extracting random 224×224 patches and their horizontal reﬂections from the 256×256 images and training our network on these extracted patches. The second form of data augmentation consists of altering the intensities of the RGB channels in training images.

They trained their models using stochastic gradient descent with a batch size of 128 examples, momentum of 0.9, and weight decay of 0.0005

They initialized the weights in each layer from a zero-mean Gaussian distribution with standard deviation 0.01. They initialized the neuron biases in the second, fourth, and ﬁfth convolutional layers, as well as in the fully-connected hidden layers, with the constant 1





**EXACT (Evolutionary Exploration of Augmenting Convolutional Topologies)**

This method, inspired by an optimization technique for ANNs called **NEAT (NeuroEvolution of Augmenting Topologies)** basically implements the same concept in NEAT for CNNs which wasn’t previously attempted due to the size and structure of CNNs and the time required to train them.

There’s the factor of computation, due to expensive training of CNNs, EXACT uses an asynchronous evolution strategy to allow scalability by a scalable distributed execution.

Convolutional Neural Networks (CNNs) have become a highly active area of research due to strong results in areas such as image classiﬁcation, video classiﬁcation, sentence classiﬁcation, and speech recognition. However less work has been made in the area of automated design of CNNs. They also use a distributed algorithm to evolve progressively more complex CNNs. This work presents a new algorithm, Evolutionary Exploration of Augmenting Convolutional Topologies (EXACT), which can evolve CNNs of arbitrary structure and ﬁlter size. Due to high computational demands, it has been implemented as part of the Citizen Science Grid1, a Berkeley Open Infrastructure for Network Computing (BOINC) volunteer computing project.

When compared to a set of human designed CNNs without max pooling and trained with the same back propagation implementation and hyper parameters, the evolved CNNs showed improved training and test data accuracy, and signiﬁcantly reduced training and testing error.

This approach is basically based on the observation that any 2 filters of any size in a CNN can be connected by a convolution of size *convd = |outd – ind|* where *out* and *in* are output filters and *conv* is the size of the convolution in dimension *d.* This allows us to evaluate CNNs solely on the basis of filter sizes and the way they are connected.

A master process manages a population of CNN genomes along with their ﬁtness. Worker processes request CNN genomes to evaluate from the master, which generates them either through applying mutation operations to a randomly selected genome in the population or by selecting two parents and performing crossover to generate a child genome. When that worker completes training the CNN, it reports the CNN along with its ﬁtness back to the master, which will insert it into the population and remove the least ﬁt genome if it would improve the population. EXACT allows for any CNN training method to be plugged in to perform the ﬁtness evaluation done by the workers. they use the MNIST handwritten digits dataset, this is a 28x28 input node, and 10 output nodes – this also happens to be the simplest benchmark NNs used to evaluate results.

Disable Edge This operation randomly selects an enabled edge in a CNN genome and disables it so that it is not used.

Enable Edge If there are any disabled edges in the CNN genome, this operation selects a disabled edge at random and enables it.

Split Edge This operation selects an enabled edge at random and disables it. It creates a new node (creating a new node innovation) and two new edges and connects the input node of the split edge to the new node, and the new node to the output node of the split edge.

Add Edge This operation selects two nodes n1 and n2 within the CNN Genome at random, such that depthn1 < depthn2 and such that there is not already an edge between those nodes in this CNN Genome, and then adds an edge from n1 to n2.

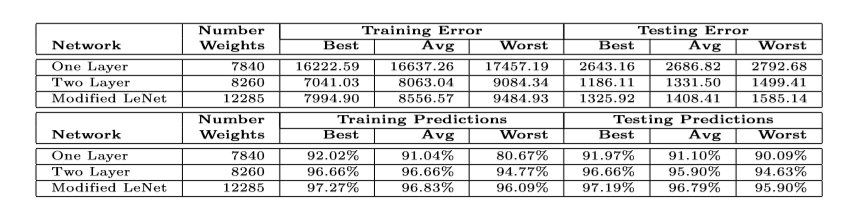
Crossover utilizes two hyper parameters, the more ﬁt parent crossover rate and the less ﬁt parent crossover rate. Two parent CNN genomes are selected, and the child CNN genome is generated from every edge that appears in both parents.

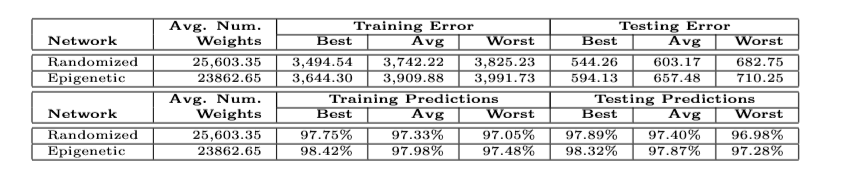
While EXACT is independent of the method used to train CNNs, it does however present an interesting opportunity for weight initialization. As after the initial population is evaluated, child genomes are generated from one or two trained parent CNNs

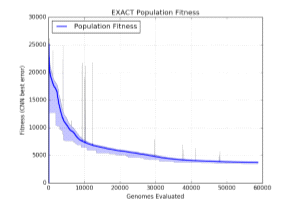
It has a multithreaded implementation for small scale use, and an MPI implementation for use on high performance computing clusters. However, to provide enough computational resources to perform EXACT on a large scale, it was also implemented as part of a BOINC project. BOINC clients running on volunteered hosts serve as worker processes, and server-side daemons were developed to validate results and handle the master EXACT process. Perhaps the most signiﬁcant technical challenge was that in order to prevent users from reporting incorrect or malicious results, each work unit needed to be sent to multiple hosts so that the results can be validated against each other.

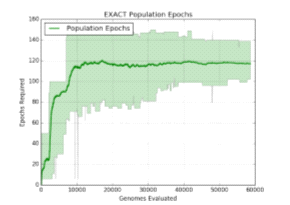
Due to issues involved in robustly developing portable binary ﬁles, check pointing of the CNNs in EXACT is done using a text ﬁle. Back propagation was done as in standard stochastic back propagation, a forward pass and backward pass with weight updates were performed once per training image, and the order of training images was shuﬄed before the beginning of each epoch.

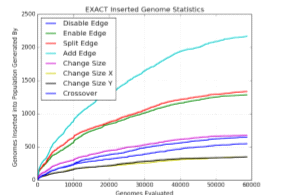
The implementation of backpropagation, its hyper parameters and possible pertaining can all have signiﬁcant eﬀects on the resulting performance of a neural network. As the focus of this work was to demonstrate the eﬀectiveness of EXACT as a evolution strategy for the structure of convolutional neural networks, and due to the fact that the EXACT algorithm can allow for the use of any neural network training strategy, three diﬀerent neural network architectures of increasing complexity were used as benchmarks to examine how well the evolved networks compare to human designed networks.











Two simultaneous EXACT searches were performed, one using epigenetic weight initialization and the other using randomized weight initialization. They both had a population size of 100, a crossover rate of 20% (which entails a mutation rate of 80%) and 3 mutation operations were performed for each genome mutated. Figures 4 and 5 show the progress of the two EXACT searches in terms of the range of ﬁtness of the populations, the range of how many epochs it took each genome to ﬁnd its minimum training error, and how many genomes were inserted into the population having been generated with crossover and the diﬀerent mutation operations.

**EFFICIENT PARALLEL LEARNING ALGORITHMS FOR NEURAL NETWORKS**

Optimizing the way neural networks learn using parallel computation techniques and mathematical models instead of back propagation have proven to be more efficient. This problem has received a lot of attention because of how these networks represent complex data mappings in an efficient parallel topology. The learning problem here, is viewed as an optimization problem, and function evaluation is very expensive, However, because the network underneath is parallel in nature, this function evaluation has been parallelized by defining the network as a function of a weight vector and an input vector which results in an output. Since this shows that the evaluation on the network function is inherently parallel, pipelining is used to evaluate multiple input vectors in constant time.

A neural network is characterized by its architecture, its node functions, and its interconnection weights .The "learning" problem for a neural network refers to the problem of finding a network function which approximates some desired "target" function TO, defined over the same set of input vectors as the network function

As they have framed it here, the learning problem is a classic problem in optimization. More specifically, network learning is a problem of function approximation, where the approximating function is a finite parameter-based system. The goal is to find a set of parameter values which minimizes a cost function, which in this case, is a measure of the error between the target function and the approximating function.

The choice of convergence criterion is important. In the case where the error surface contains "bad" local minima, it is possible that the error threshold will be unattainable, and in this case the algorithm will never terminate. Unfortunately, for practical problems where this limit is not known a priori, this approach is inapplicable.

Steepest Descent is the most classical gradient-based optimization algorithm. In this algorithm the search direction d" is always the negative of the gradient - the direction of steepest descent.

There are many ways to perform this computation, but they are all iterative in nature. They are not interested in such techniques because they are less parallelizable than the methods they have pursued and because they are more expensive, both computationally and in terms of storage requirements. Because they are implementing our algorithms on the Connection Machine, where memory is extremely limited, this last concern is of special importance.

Conjugate gradient techniques take advantage of second order information to avoid the problem of cross-stitching without requiring the estimation and storage of the Hessian.

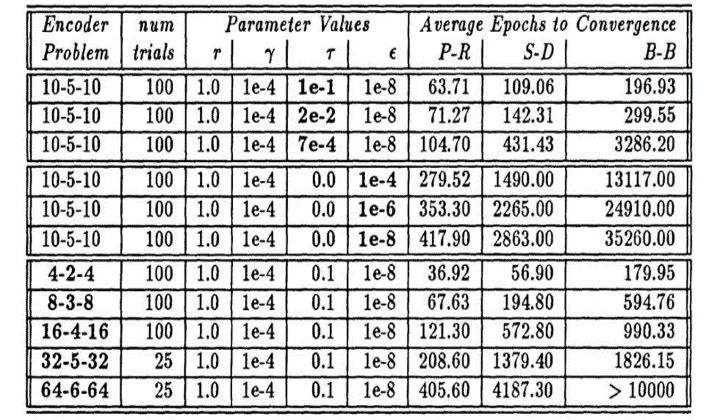
The Batch Back-propagation algorithm can be described in terms of our optimization framework. Without momentum, the algorithm is very similar to the method of Steepest Descent.

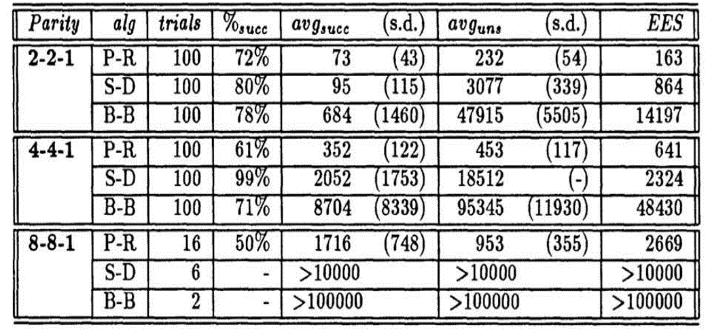
One of the problems of performing gradient descent on the "error surface" is that minima may be at infinity. Thus an algorithm may have to travel a great distance through weight space before it converges. Many researchers have found that weight decay is useful for reducing learning times this technique can be viewed as adding a term corresponding to the length of the weight vector to the cost function; this modifies the cost surface in a way that bounds all the minima.

They have emphasized the parallelism inherent in the evaluation of EO and gO. To be efficient, any learning algorithm must exploit this parallelism. Without momentum, the Back-propagation algorithm is the simplest gradient descent technique, as it requires the storage of only a single vector.

The additional computational requirements are essentially those needed for line search - a single dot product and a single broadcast per iteration. These operations are parallelizable so the additional computation required by these algorithms is also minimal, especially since computation time is dominated by the evaluation of EO and gO. Both the Steepest Descent and Polak-Ribiere algorithms are easily parallelizable.

We have compared the performance of the Polak-Ribiere (P-R), Steepest Descent (S-D), and Batch Back-propagation (B-B) algorithms on small Boolean learning problems. In all cases we have found the Polak-Ribiere algorithm to be significantly more efficient than the others.

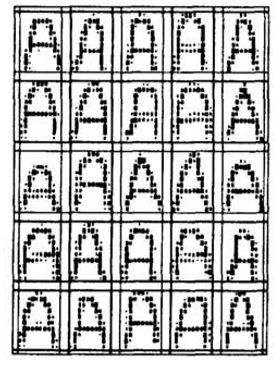


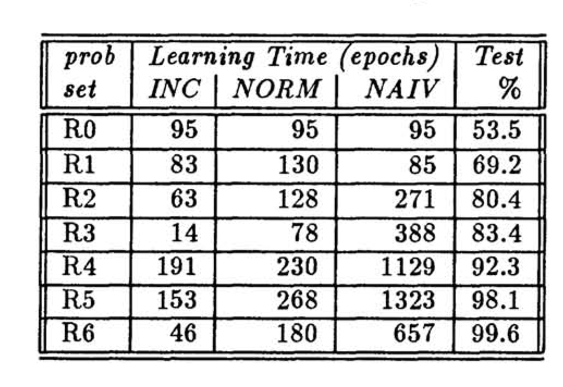


Characters were hand-entered in a 80 x 120 pixel window with a 5 pixel-wide brush (mouse controlled).

A training set consisting of 64 distinct sets of the 26 upper case letters was created by hand in the manner described.25" A" vectors are shown in figure 1. This large training set was recursively split in half to define a series of 6 successively larger training sets. A testing set consisting of 10 more sets of hand-entered characters was also created to measure network performance.

Both Polak-Ribiere and On-line Back-propagation were tried on all problems. Table 3 contains only results for the Polak-Ribiere method because no combination of weight-decay and learning rate were found for which Back-propagation could find a solution after 1000 times the number of iterations taken by Polak-Ribiere





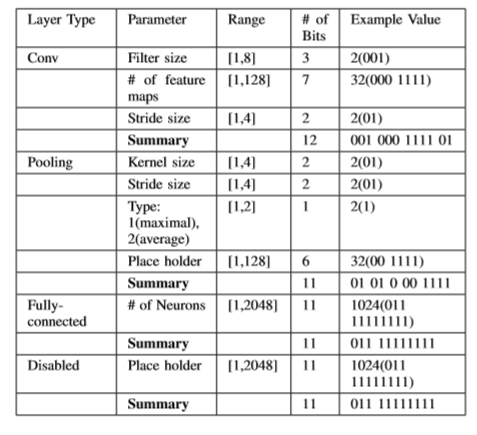
In addition, this algorithm is well-suited to parallel implementation on massively parallel computers such as the Connection Machine. Finally, incremental learning is a way to increase the efficiency of optimization techniques when applied to large real-world learning problems such as that of handwritten character recognition.

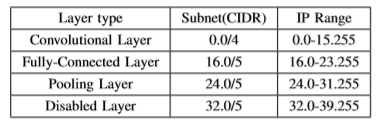
Evolving Deep Convolutional Neural Networks by Variable-length Particle Swarm Optimization for Image Classification

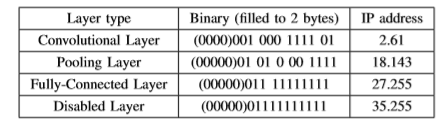
This method uses the nature-inspired meta-heuristic PSO (Particle-swarm optimization) to automatically search for the optimal architecture of CNNs. However, the learning process for large data is too slow due to the high computational cost for most of the methods and it might not be practical for industrial use.

The overall goal of this paper is to design and develop an effective and efﬁcient PSO method to automatically discover good architectures of CNNs.

In terms of the conﬁguration, apart from the output layer which will almost be a number of neurons having the number of classes in the classiﬁcation problem as the size, different types of layers have different conﬁgurations. A strategy is encoded in the particle-vectors of PSO which allows them to easily encode CNN layers. PSO is a population-based algorithm motivated by the social behaviour of fish schooling or bird flocking commonly used for optimization problems without enough requisite domain knowledge. This algorithm generates a population of solutions that traverse the solution space to find the best solution by updating their velocity and position vector. The issue is that the population of solutions generated would all be representing the same dimensions of data which would not be able to predict variable-length architectures for CNNs. Here, based on how IP addresses work to differentiate between different devices connected to a network, the encoded binary that represents the combination of the IP address and the subnet mask is used for network identification. Using the former, we can have each architecture variable value, encode it to binary and fuse them together as mentioned to create one large binary string. However, this might lead to a very horrendous search time for each PSO solution particle, therefore, as IP addresses are divided into parts using decimal points, we would divide the binary string into parts and each part would be converted to bytes which would then be used to encode the PSO solution particles to various layer architectures in a CNN. Now to encounter the variable-length architectures of CNNs, an efficient way of disabling layers in the architecture would be designed and encoded in the solution particle. Due to expensive training times, partial datasets are used for evaluation to speed-up the process.



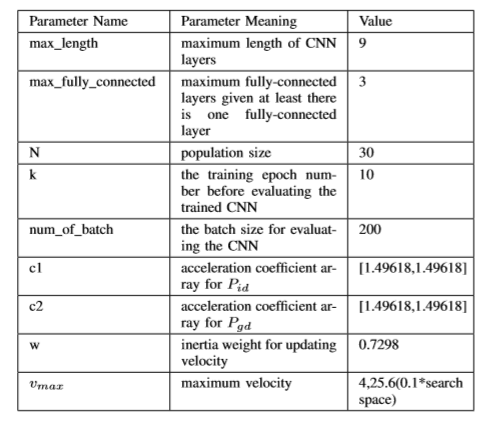




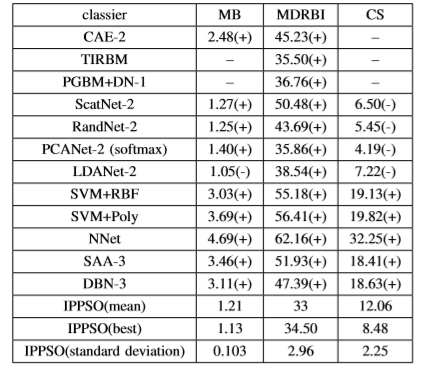
Before performing the ﬁtness evaluation, a proper weight initialisation method has to be chosen, and Xavier weight initialisation is chosen as it has been proved as an effective way, and has been implemented in most of Deep Learning frameworks.

In these experiments, three datasets are chosen from the widely used image classiﬁcation benchmark datasets to examine the performance of the proposed IPPSO method. The ﬁrst two benchmark datasets are two of the MNIST variants for classifying 10 hand-written digits (i.e., 0-9). The third benchmark dataset is for recognizing the shapes of objects which contains 8,000 training images and 50,000 test images.

In the experiments, state-of-the-art algorithms, that have reported promising classiﬁcation errors on the chosen benchmarks, are collected as the peer competitors of the proposed IPPSO method



It is clearly shown in Table V that by comparing the mean classiﬁcation errors of the proposed IPPSO method with the best performance of the peer competitors, IPPSO performs the second best on the MB dataset, which is only a little bit worse than LDANet-2.



The goal of this paper was to develop a new PSO approach with variable length to automatically evolve the architectures of CNNs for image classiﬁcation problems. In this paper, we have investigated the proposed IPPSO method for evolving deep CNN and it is proved of obtaining promising results. Based on this research, there are a couple of further researches that are worth doing.

CGP

This technique uses CGP (Cartesian genetic programming) to automatically encode the CNN architecture for an image dataset. The CNN connectivity represented by CGP encoding is optimized to maximize the validation accuracy. Validation accuracy arises from the use of a validation dataset after the training dataset.

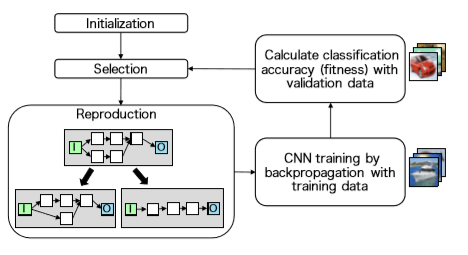
CGP (Cartesian genetic programming) is a form of genetic programming which uses a graph representation to encode the CNN. This approach allows encoding for variable-length architectures for CNNs and also allow for the implementation of shortcut-connections.

They can consider neural network architecture design as the model selection or hyper parameter optimization problem from a machine learning perspective. .Naturally, evolutionary algorithms have also been applied to hyper parameter optimization problems .The hyper parameter optimization approach often tunes predefined hyper parameters, such as the numbers of layers and neurons, and the type of activation functions. Although this method has seen success, it is hard to design more flexible architectures from scratch.

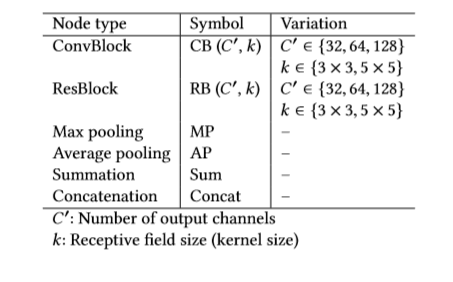
The DPPN is a differentiable version of the compositional pattern-producing networks(CPPNs)[32].This paper focuses on the effectiveness of in direct coding for weight optimization. That is, the general structure of the network should be predefined.

Interesting approaches ,including the automatic designing of the deep neural network architecture using reinforcement learning, were attempted recently.

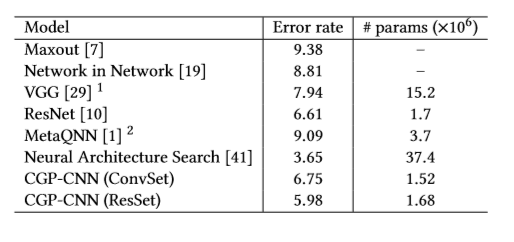
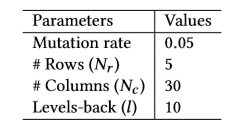
They use the CGP encoding scheme, representing the program as directed acyclic graphs with a two-dimensional grid defined on computational nodes ,for the CNN architecture representation.



The fitness evaluation of the CNN architectures is so expensive because it requires the training of CNN. To efficiently use the computational resource, they want to evaluate some candidate solutions in parallel at each generation.



They consider two experimental scenarios: the default scenario and the small-data scenario. .The default scenario uses the default numbers of the training images, where as the small-data scenario assumes that we use only 5,000 images as the learning data.



We compare the classification performance of our method with the state-of-the-art methods and summarize the classification error ratesinTable3 .We refer to the architecture constructed by the proposed method as CGP-CNN. In the small-data scenario ,they compare our method with VGG and ResNet. We have trained VGG and ResNet models by the same setting of the re-training method in the proposed method; it is based on the training method of the ResNet.

However ,our proposed method requires much computational cost; the experiment on the default scenario needed about a few weeks in our machine resources. We can reduce the computational time if the training data are small .Thus, one direction of future work is to develop the evolutionary algorithm to reduce the computational cost of the architecture design, e.g., increasing the training data for the neural network as the generation progresses.

NICO (Nature-inspired CNN Optimizer)

Methodology

We define the CNN framework as a mathematical model i.e. as a network function N (w, r) = O(w) where w is the weight vector for a neural layer and r is the input of that neural layer and O(w) is the output vector provided by that layer [4].We define the search space of our evolutionary algorithm as an n-dimensional graph where each point represents an n-dimensional vector that contains the parameters that the user wants to be optimized for a dataset. The “n” is user-defined and can be manipulated based on the user’s will.We define a **master-slave distributed architecture** for training and evaluating our CNNs to introduce parallelization to make the evaluation process faster and more efficient.

The objective function of our evolutionary algorithm is complex and requires heavy computation. Therefore, we would use **dynamic programming** to avoid repeated calculation of the objective function.Each session of the slave systems will be logged to observe and keep track of the performance and progress.

Compare And Contrast

We attempt to fuse and hybridize approaches that we’ve reviewed in detail above into a single culmination of an efficient automation of CNN design. While previous papers do not attempt to approach the optimization of all CNN parameters at the same time, we propose to target all parameters simultaneously and employ generic techniques to make the automation process more efficient while staying in computational limits.