Nature Inspired CNN Topology Optimization

# **TEAM**

|  |  |
| --- | --- |
| **Name** | **Roll Number** |
| Mohsin Ashraf | 15140094 |
| Mushood Hanif | 14140062 |
| Mehwish Hameed | 15140107 |
| Rida Yaqoob | 14140044 |

Contents

[**TEAM** 1](#_Toc5717664)

[**Problem Statement** 3](#_Toc5717665)

[**Introduction** 3](#_Toc5717666)

[**Related Work** 3](#_Toc5717667)

[**EXACT (Evolutionary Exploration of Augmenting Convolutional Topologies)** 3](#_Toc5717668)

[**Evolving Deep Convolutional Neural Networks by Variable-length Particle Swarm Optimization for Image Classification** 3](#_Toc5717669)

[**ImageNet** 4](#_Toc5717670)

[**Efficient Parallel Learning Algoithms For Neural Networks** 4](#_Toc5717671)

[**NICO (Nature-inspired CNN Optimizer)** 4](#_Toc5717672)

[**References** 5](#_Toc5717673)

# **Problem Statement**

The **CNN Framework** has been proven many a times for its prowess and degree of performance on image datasets [1]. However, CNNs are considered as **deep learning networks** and require heavy computation, also, to find the optimal **topology** to initialize CNNs for any dataset still remains a problem.

# **Introduction**

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;

## **Related Work**

### **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.

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.

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.

### **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. There are basically 3 steps involved for achieving this optimization:

* 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.

### **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.

### **Efficient Parallel Learning Algoithms For Neural Networks**

Optimizing the way neural networks learn using parallel computation techniques and mathematical models instead of backpropagation 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.

### **CGP**

### **NICO (Nature-inspired CNN Optimizer)**

* 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.

# **References**

1 - ImageNet Classification with Deep ConvolutionalNeural Networks – Hinton, University of Toronto

2 – Large Scale Evolution of Convolutional Neural Networks Using Volunteer Computing – Travis Desell

3 - Evolving Deep Convolutional Neural Networks by Variable-length Particle Swarm Optimization for Image Classification – Bin Wang, Victoria University of Wellington

4 - Efficient parallel learning algorithms for neural networks – Alan Kramer, U.C. Berkeley