Efficiently Coevolving Deep Neural Networks and Data Augmentations

Shane Acton, Sasha Abramowitz, Liron Toledo, Geoff Nitschke
Computer Science Department
University of Cape Town, South Africa
actsha001@myuct.ac.za, abrsas002@myuct.ac.za, tldlir001@myuct.ac.za, gnitschke@cs.uct.ac.za

***Abstract*—Designing large *deep learning neural networks* by hand requires tuning large sets of method parameters, requiring trial and error testing and domain specific knowledge. Neuro-evolution methods such as *CoDeepNeat* (CDN), based on *Neuro-evolution of Augmenting Topologies* (NEAT), apply evolutionary algorithms to automate deep neural network parameter optimisation. This paper presents and demonstrates various novel beneficial extensions to the CDN method, including new genotypic speciation mechanisms, special mappings in deep neural network encodings, as well as evolving *Data Augmentation* schemes. Results indicate that these CDN method variants yield significant task-performance benefits over the benchmark CDN method when evaluated on a popular public image recognition data-set.**

I. Introduction

Motivated and enabled by significant improvements in computational processing power, advances in multi-core and high-performance computing architectures, and increasing availability of high-dimensional data-sets, *deep-learning* [1] has recently achieved state-of-the-art results in various traditional machine-learning benchmark applications. This includes significantly task performance gains over previous machine-learning methods in image recognition [2], speech recognition [3], game-playing [4], and drug discovery [5]. Generally, deep artificial neural networks coupled with suitable supervised learning mechanisms and sufficient compute-time and computational resources have consistently demonstrated an effective capacity to generalise from patterns learnt across a vast range of high-dimensional complex data-sets [1].

A *Convolutional Neural Network* (CNN) is a deep learning architecture primarily used for computer vision tasks [6]. In traditional CNNs, images are initially passed through *convolutional* layers [7], which extract increasingly abstract features from images. After feature extraction, the convolutional layer outputs are passed into fully connected layers which learn a target function [1]. CNNs currently yield state-of-the-art task performance across a range of image recognition tasks using large benchmark image data-sets such as *ImageNet* [8].

Deep learning algorithms using CNNs usually require very large, well labeled and annotated data-sets in order to yield comparable or superior task performance, when compared to other machine learning methods for various applications [1]. Attaining such data-sets has been a key challenge in demonstrating the efficacy of deep-learning algorithms for various tasks [9], and this is where *Data Augmentation* (DA) has become indispensable [10]. DA is a regularisation technique that artificially inflates a training data-set by performing label-preserving transformations to add more uniform examples. A lack of quality labeled training data is a major cause of over-fitting [11], and DA has been demonstrated as effective means of synthetically augmenting training data to mitigate over-fitting and boost CNN task-performance [10].

However, combinatorial space of possible DA operations is extremely large for complex CNN training tasks and ensuring correct labelled transformations is difficult and error-prone. For example, vertically flipping images of text and numbers results in meaningless data. Additional research has indicated that finding a suitable DA method is also dependant on the CNN architecture [12], [13], thus further complicating manual selection of an appropriate DA method.

State-of-the-art deep learning artificial neural networks are becoming increasingly sophisticated, have many parameters and rely on experts with specific application domain knowledge for designing neural network architectures and tuning methodological and experimental parameters [14]. Finding optimal neural architectures and associated learning and experiment parameters is usually time consuming. It requires skilled human experimenters to run various tuning tests for neural architecture, learning and experiment parameters in order to intuit suitable combinations of parameters and architecture designs for any given task. To alleviate such design and parameter tuning problems, *Automatic Machine Learning* (AutoML) [15], has been proposed to automate the design of optimal neural architectures for given tasks. For example, AutoML has been effectively demonstrated for largely automating neural network topology design and parameter selection for artificial neural network image classifiers [16], automating training data pre-processing for such classifiers [17], as well as automating bounds and constraints placed on the search domain in order to increase classification efficacy [18].

Currently, there is no canonical or established approach to the methodological design of the AutoML algorithms themselves. Use of techniques such as *grid-search* [19] become prohibitively expensive in terms of time and computational resources for complex tasks. However, recent AutoML methods have applied *Evolutionary Algorithms* (EAs) [20] to *evolve* (automate the search for) optimal neural network topology and associated learning parameters [21].

This is a branch of EA research popularly known as *Neuro-Evolution* (NE) [22]. *Neuro-Evolution of Augmenting Topologies* (NEAT) [23] uses direct encoding neuro-evolution to evolve neural network connection weights and topologies. NEAT applies three key techniques to enable the evolution of effective and efficient neural network solutions. First, NEAT assigns a unique historical marking to every new neuron (gene) so that crossover can only be performed between pairs of matching genes.

Second, NEAT speciates the population so as networks (genotypes) compete primarily within their own niches (identified by historical markings) instead of competing with the whole population. Third, NEAT begins evolution with a population of simple networks with no hidden nodes but gradually adds new topological structure (nodes and connections) using specialised mutation operators: *add hidden node* and *add link*. An advantage of this NEAT *complexification* process is that it will likely find a solution in lower dimension search spaces compared to relatively large search spaces corresponding to large fixed topology networks specified *a priori*. However, direct encoding and *complexification* also means it is unlikely that NEAT will evolve deep neural networks amenable for solving various complex classification and recognition tasks.

The AutoML method *CoDeepNEAT* (CDN) [24], [25] applies NEAT to automatically adapt deep neural network topology, while concurrently training the networks with supervised learning and given data-sets. CDN has been demonstrated as achieving near-optimal task performance in some text and images classification tasks [25], though is currently unable to achieve such near-optimal task performance on popular benchmark image recognition data-sets such as CIFAR-10[1].

This paper's main objective is to demonstrate the efficacy of new neuro-evolution operator and evolutionary data augmentation extensions to the CDN method (section III), with the goal of boosting overall task-performance of evolving deep neural network architectures. Thus, the main contribution of this study is the formulation of these methodological extensions for CDN (section III), and the demonstration of the efficacy of these CDN method variants in comparative experimental evaluations (section IV) on the CIFAR-10 data-set.

II. Methods I: CoDeepNEAT (CDN)

CoDeepNEAT (CDN) is an extension of NEAT designed for deep-learning [23]. Both NEAT and CDN evolve neural network topological structure however, instead of evolving weight values, as in NEAT, CDN uses *backpropagation* [6] to find the weights of relatively large *Convolutional Neural Networks* (CNNs). CNN task performance accuracy on a given data-set is used to comparatively score a population of CNNs and thus determine which CNNs are selected as parents for application of evolutionary operators [20], and thus propagate the next generation (population) of CNNs.

CDN combines unsupervised (evolutionary) and supervised (backpropagation) learning optimize the topological size (number of parameters) and task performance (classification accuracy) of CNNs. It achieves this by using a multi-objective EA to find the *Pareto* front [26] when ranking CNNs in the population. CDN used multi-objective optimisation in order to select CNNs that have suitable trade-offs of effectiveness (high task-performance) and computational efficiency (minimal topological size) [24]. To enable the evolution of effective and efficient CNNs, the CDN method *co-evolves* two genotype populations: *blueprints* and *modules*.

Modules represent repeatable, independently functional neural network substructures, assembled to form complete CNNs, where each node (gene) in a module genotype represents one DNN layer and its parameters (for example, layer type, layer size, kernel size and activation function). *Blueprints* specify how these repeating structures (modules) are connected in order to form a complete CNN. To achieve this, nodes in a blueprint genotype sample modules and connect them.

The module population is decomposed into multiple subpopulations (species) derived using the NEAT *speciation* mechanism [23]. Blueprint nodes contain references to module species, and each node randomly samples a module linked to its species (Figure 1). When multiple blueprint nodes sample the same species, all such nodes are assigned the same module from a given species. This promotes repeated substructures in the evolved CNN, which in turn promotes increased effectiveness and efficiency in neural processing [24], [25].

For CNN population evaluation at each CDN generation, every blueprint genotype is parsed into the assembly of multiple distinct CNNs (as a blueprint node is likely to sample distinct modules given each genotype parsing) [24], [25]. After a given number of generations of evolution, the fittest (highest classification accuracy) CNNs are selected for further training on the given data-set, with the goal of finding the maximum classification accuracy. CDN also uses *Data Augmentation* (DA) [10] to complement data-sets and improve accuracy on training and test data. The authors of CDN constructed a DA scheme where each blueprint performed the same augmentation operations in the same order. However, each blueprint evolved its own DA scheme parameters to enable overall increased task-performance (accuracy) for evolved CNNs [24].

III. Methods II: CDN Extensions

To address our objective of demonstrating the efficacy of new methodological extensions to *CoDeepNEAT* [24], [25] (CDN, section I), we present a complete re-implementation of CDN, *base-CDN*[2] and two CDN method variants: MMS-CDN: *CoDeepNEAT with ModMax and Speciation* (sections III-A

[1]https://www.cs.toronto.edu/~kriz/cifar.html

[2]https://github.com/sash-a/CoDeepNEAT

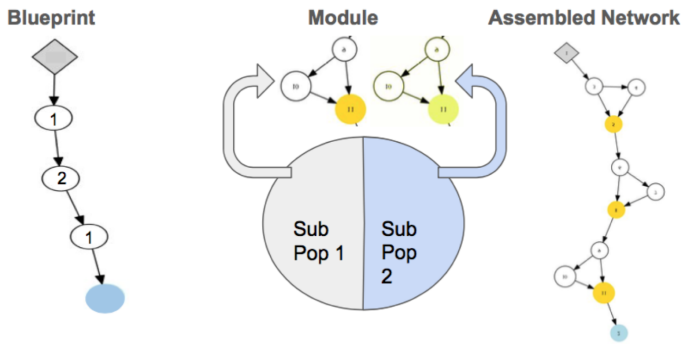

Fig. 1: An example [24] of how *modules* (neural network sub-structures) are described by *blueprints* (module connectivity map) in CoDeepNEAT. Numbers in the blueprint (left) genotype refer to a module (center) sub-population (species). The neural network (right) is assembled via using each blueprint node corresponding to a specific module sampled from its given species.

and III-B), and MMS-DA-CDN: *CoDeepNEAT with ModMax, Speciation and Data Augmentation* (section III-C).

MMS-CDN includes an extension (*ModMax*, section III-A) to improve genotype to neural network architecture mapping, and an extension to preserve effective functional specialization that emerges in sub-populations (species) of modules (*Speciation*, section III-B). Whereas, MMS-DA-CDN is a novel method for the automating the data augmentation process of CDN, via evolving data augmentations schemes suitable for the given training data (section III-C).

A. MMS-CDN: Component I: ModMax

ModMax is the first component of the CDN variant: MMS-CDN. The second: *Speciation*, is described in section III-B.

1) Module Retention: Enables evolutionary *elitist* selection [20], via having the best performing (highest accuracy) blueprints maintain references to modules sampled in previous generations. In *base-CDN* blueprints randomly sample their modules at the beginning of each generation, thus it is unlikely that the best CNNs would be recreated in later generations, even if their constituent components survive. This makes *base-CDN non-elitist*, as successful CNNs are not guaranteed to survive into later generations. Module retention attempts to solve this issue.

Module retention directly encodes sampled modules in the blueprint genome, ensuring the parsing from blueprint to CNN can be deterministic, and can thus be subject to elitist selection. Specifically, module retention stores the modules that blueprint nodes sample inside the blueprint's genotype.

The mapping from module species (sub-populations) to module individuals (neural network sub-structures) is defined for a blueprint (B_b) as a sample mapping (SM_b). Each parse (p) of a blueprint (B_b) entails a mapping from a module species to a specific module (SM_b^p). Sample mappings (SM_b) are also subject to crossover and mutation (Table II) and also improved with elitist selection across generations.

This prevents (non-elitist) selection or assembly of weak blueprints (resulting from random module sampling), as observed when running *base-CDN* (Figure 2), and also promotes the propagation of high task-performance (accuracy) CNNs over successive generations.

2) Maximum Fitness Aggregation: Since blueprints and modules are evaluated multiple times per generation (section III-A1), we aggregate fitness via taking the highest fitness (per generation) for each blueprint and module. This maximum fitness aggregation means that modules with specific functions are not be penalised for being used by low fitness blueprints (as would be the case in mean fitness aggregation). For example, if a module is used as an *output* node function of a CNN, but referenced by a blueprint as an *input* node function of the CNN, then this would result in the blueprint being evaluated with a low fitness. However, the maximum fitness aggregation allows a module which has been used incorrectly to survive as long as it has been used correctly at least once. Thus, maximum fitness aggregation, in combination with Module Retention ensures the best CNNs are guaranteed to be reconstructed in subsequent generations.

B. MMS-CDN: Component II: Speciation Extensions

All CDN method variants use a module speciation mechanism similar to NEAT [23], that operates in the module genotype space, and provides a computationally cheap proxy for measuring functional similarity between modules (neural network sub-structures). At the end of each generation, a new representative (module) is randomly selected from each species. This is then the module used for similarity comparisons between that given species and other modules.

In this study's experiments (section IV), at each generation, modules are placed in the species whose representative they are most similar to (according to a topological similarity metric [23]). Ideally, all modules in a species will serve a similar function, so replacing any module reference in a blueprint with another module reference from the same species will not significantly impact task-performance of the CNN (encoded by

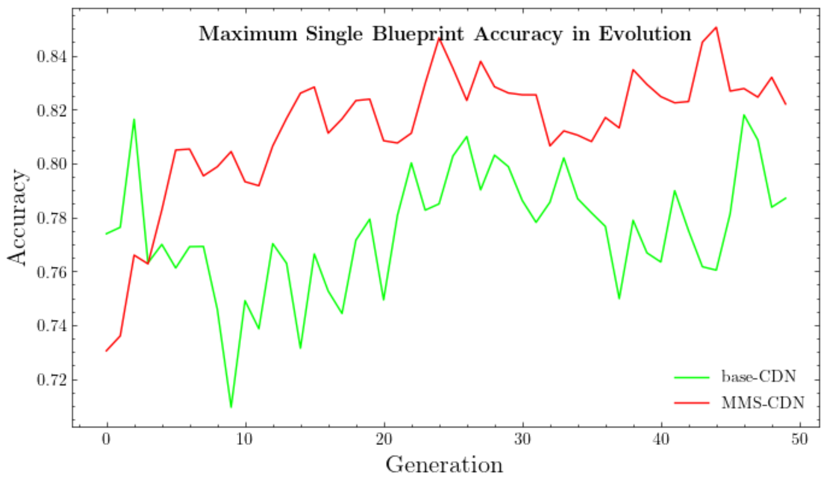

Fig. 2: Maximum blueprint accuracy (normalised to the range: [0.0, 1.0]) at each generation of neuro-evolution for a *base-CDN* and *MMS-CDN* experiment. Note, *base-CDN* (compared to MMS-CDN) is prone to up to 10% drops in accuracy (green line).

all modules references in the CNN blueprint). The following discusses this further.

1) Centroid Representative Selection: Speciation of modules (neural network sub-structures) enables the evolution of functional specialization within different parts of the network, where such specializations complement each other and boost overall network task performance [27]. For example, one species may specialize to evolving network output layer modules. However, a species may also contain several *outlier modules* (modules specialised to different sub-network functionality). In this case random representative selection could result in an outlier module being selected as the representative. At each generation, centroid speciation prevents this via selecting the individual (module) with the maximum similarity to all other individuals in the species (the *centroid representative*), rather than a random representative. Specifically, given species $S = \{I_i \mid i \in [0..n)\}$, where I_i is the i^{th} member of S and n is the number of individuals, the average similarity of each individual I_i to all other members of that species is:

$$avgSim(i) = \frac{1}{n}\sum_{j=0}^{n} similarity(I_i, I_j)$$

We then compute the representative of the species as the individual with the maximum average similarity:

$$representative = max_i(\{avgSim(i) \mid i \in [0..n)\})$$

This promotes modules within each species to remain functionally similar while species themselves tend to be dissimilar yet complementary in terms of network functionality.

C. Data Augmentation Evolution

This section describes an evolutionary *Data Augmentation* (DA) scheme[3] that enables further improvements to the supervised learning component of the CDN method (section II), via evolving new data augmentations [10] and thus enhancing the quality of CNN training. Previous AutoML algorithms have used recurrent neural network controllers [12] and generative networks [13] to predict suitable data augmentation policies for given CNN architectures and data-sets. However, our DA-CDN variant enables the co-evolution of data-augmentation schemes together with evolving blueprints and modules for the automated derivation of CNN topologies that work best with a given data-set (section II).

The DA-CDN method variant thus cooperatively evolves DA schemes (within the population of blueprints) that in turn encode CNN topologies optimised for a given data-set. DA-CDN is beneficial over previous automated DA schemes [12], [13], as it reduces the need for user-specified domain-specific knowledge, is comparatively computationally cheap, and also optimises the order in which DA operations are executed to boost overall CNN supervised learning performance.

DA genotypes (Figure 3) are encoded as part of each blueprint genotype, which allows the fittest blueprints and their corresponding DA schemes to be selected for during the evolutionary process. DA genotypes (schemes) corresponding to each blueprint are selected from a separate population (equal size to the blueprint population, Table II), where both the DA scheme and blueprint are subject to mutation and crossover operators (Table II).

Each DA genotype is represented as a linear directed graph, with no branching (Figure 3), where each node indicates a separate DA operation (Table I), and the entire graph represents the *DA scheme*. DA-CDN initialises blueprints with DA schemes consisting of two random nodes (DA operations). Such DA operations are randomly selected from a predefined set of DA operations (Table I), and DA schemes can be mutated to any number of nodes during neuro-evolution. This allows for a larger exploration space of DA schemes when compared to the system used in original CDN.

[3]Data Augmentation (DA) library: *imgaug*: https://github.com/aleju/imgaug

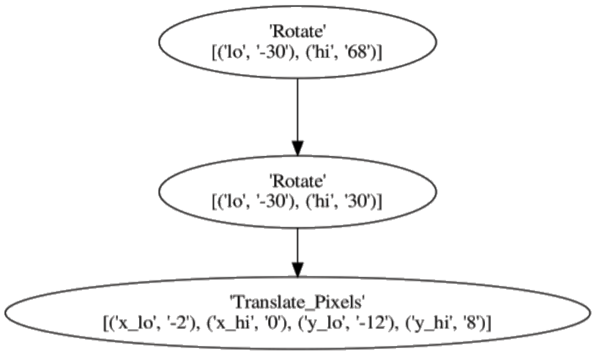
'Rotate'
[('lo', '-30'), ('hi', '68')]
'Rotate'
[('lo', '-30'), ('hi', '30')]
'Translate_Pixels'
[('x_lo', '-2'), ('x_hi', '0'), ('y_lo', '-12'), ('y_hi', '8')]

Fig. 3: An example *Data Augmentation* (DA) genotype. This genotype would rotate an image twice and then translate it. The value range for these operations are below the operation name, where, exact values used are randomly sampled from this range.

DA Operator	**Operator Description**
Flip lr	Flips image horizontally
Translate Pixels	Translate image along the x-axis or y-axis
Rotate	Rotates image
Scale	Scale image to percentage of its original size in either its x or y dimensions
Pad Pixels	Adds rows or columns of blank pixels to image
Crop Pixels	Removes existing rows or columns from image
Coarse Dropout	Randomly erases portions of an image
Grayscale	Manipulates alpha value of image
HSV	Manipulate image's Hue (H), Saturation (S) or Lightness Value (V)

TABLE I: *Data Augmentation* (DA) operations applied during DA evolution of the training data-set (section III-C)

IV. Experiments and Results

Three experiment sets were executed[4], one for each CDN method variant: *base-CDN*, *MMS-CDN* and *MMS-DA-CDN* (Table IV). Each experiment set comprised of 50 generations of neuro-evolution together with 8 epochs of training per CNN evaluated, and finally a further 150 epochs of training to convergence for the best CNNs (Table II).

All training was done using the CIFAR-10 image data-set, which was also used for the original CDN experiments [24], [25]. Each CDN variant experiment was executed for 10 runs to control for stochastic variation in accuracy between runs.

Figure 4 presents average maximum and average CNN accuracy results, for each CDN variant (base-CDN, MMS-CDN, MMS-DA-CDN), given the experiment and method parameters specified in Table II. For each experiment set (10 runs) we selected the five (5) fittest evolved CNNs after 50 generations of neuro-evolution, and trained each to convergence (no change in CNN accuracy given successive epochs). Given task-performance boosts demonstrated in previous evolutionary AutoML methods [24], [28] we enlarged the size of the CNN before training to convergence.

This entailed multiplying the size of each CNN layer by a constant factor (1, 3, 5), to increase its size and thus classification efficacy. We call this *feature multiplication*, observing that typically a feature multiplication value of 5 achieved the highest accuracy overall, and parameter tuning indicated diminishing task-performance benefits for higher feature multiplication values. That is, parameter tuning and preliminary testing demonstrated only a 0.4% accuracy increase by changing the feature multiplier value from 5 to 20.

[4]Each experiment run used up to four Tesla v100 GPUs on a computing cluster. An overview of all experiment and method parameters and computational statistics is available at: https://github.com/sash-a/CoDeepNEAT/

V. Discussion

Table III presents the maximum and average results for the original CDN implementation (CDN, section II), and each CDN method variant (base-CDN, MMS-CDN, MMS-DA-CDN), where all CNNs were evolved and trained on the CIFAR-10 data-set. Note, only one run (n=1) was reported in the original CDN study [24], [25]. Hence, 92.7% is presented for both maximum and average accuracy (Table III).

Given the limited results reported in the original CDN study [24], [25], we focus our discussion on our CDN re-implementation (base-CDN), and our CDN variants:

Experiment Parameter	Value	Description
Experiment runs	10	The number of replications per experiment
Generations	50	The number of generations per CDN variant run
Epochs in evolution	8	The number of training epochs during evolution
Epochs in convergence	150	The number of training epochs after evolution when fully training networks
Evaluations per blueprint	4	Number of times each blueprint is evaluated each generation
Module population size	50	Number of modules per generation
Blueprint population size	20	Number of blueprints per generation
DA population size	20	Number of DA schemes per generation
Module species size	4	Number of module species *groups*
Blueprint add node chance	0.16	Probability to add a node to a blueprint genotype
Blueprint add connection chance	0.12	Probability to add a connection node to a blueprint genotype
Blueprint node species switch chance	0.15	The chance for a blueprint node to change its module species link
Module add node chance	0.1	Probability to add a node to a module genotype
Module add connection chance	0.1	Probability to add a connection node to a blueprint genotype
Module layer type switch chance	0.1	Probability a module node will change its layer type (e.g., *Convolutional* to *Linear* or *Regulariser*)
Fitness aggregation	[max, avg]	Function combining fitness values from multiple *module*, *blueprint* and DA scheme evaluations per generation (section III-A2)

TABLE II: Experiment and CDN method variants parameters. These parameters are most relevant to experiments presented here. A complete list of parameters, values, and descriptions can be found online[4].

Method Variant Name	Runs (n)	Maximum Accuracy (%)	Average Accuracy (%)
CoDeepNeat (CDN)	1	92.7	92.7
CoDeepNeat Re-implementation (base-CDN)	10	90.1	85.8
CoDeepNeat with ModMax, Speciation (MMS-CDN)	10	91.1	87.8
CoDeepNeat with ModMax, Speciation and Data Augmentation (MMS-DA-CDN)	10	92.8	88.6
MMS-DA-CDN : 25 epochs	3	94.5	92.8

TABLE III: Maximum and average task-performance (accuracy), over 10 runs for each CDN method variant (base-CDN, MMS-CDN, MMS-DA-CDN), and the original CDN implementation, given a *feature multiplication* factor of 5 (section IV).

MMS-CDN, and MMS-DA-CDN. Hence, we applied *Mann–Whitney U* tests [29] to gauge significant differences between comparative task-performance (accuracy) results of base-CDN, MMS-CDN, and MMS-DA-CDN.

Each results set was $n = 50$, attained via selecting the 5 fittest individuals (blueprints) from each of the 10 runs for each CDN method variant (Table III). Comparing CDN to our re-implementation (base-CDN) one may note the discrepancy between the original CDN results (92.7%, [24], [25]), versus our base-CDN results (90.43% with a feature multiplication value of 3, section II).

This is attributed to differing method parameter values which were necessarily tuned for our experiments (Table II), since pertinent parameter tuning and implementation details were not reported in the CDN study [24], [25]. For example, large variations in CNN accuracy were observed via varying the *merge layers* [24] parameter.

However, all statistical comparisons discussed here are for a *feature multiplier* of 5, given that this yielded the most favourable results across all CDN variants (section IV).

First, statistical tests were applied for pair-wise (method accuracy) comparisons between base-CDN and MMS-CDN, and base-CDN and MMS-DA-CDN. *Mann–Whitney U* tests indicated the average maximum (Figure 4, left) and average (Figure 4, right) accuracy of MMS-CDN was significantly higher than base-CDN ($p < 0.01$). Similarly, the average maximum and average accuracy (Figure 4) of MMS-DA-CDN was significantly higher than base-CDN ($p < 0.01$).

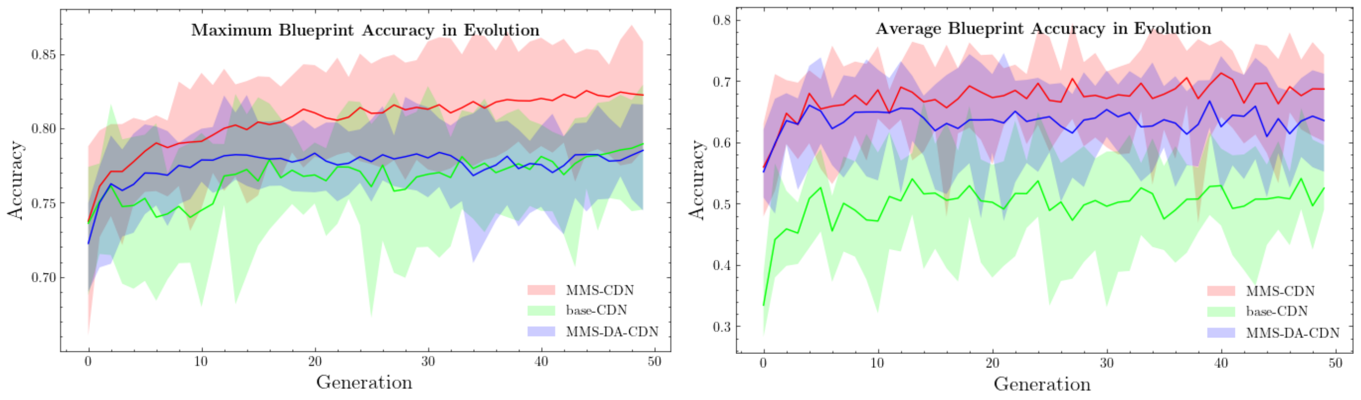

Fig. 4: Average maximum (left) and average (right) task-performance (accuracy) of CDN variants: *MMS-DA-CDN*, *MMS-CDN* and *base-CDN* (CDN), achieved by blueprints per generation (per variant). Color bands present maximum and minimum accuracy across generations per variant (solid lines indicate the average). Note differing accuracy scales (left versus right).

We theorize this to be a result of the *module retention* and *maximum fitness aggregation* elitism extensions in the *ModMax* component of MSS-CDN (section III-A). This enabled MMS-CDN to consistently evolve fitter solutions overall (compared to base-CDN, Figure 4). This gave MMS-CDN a more suitable CNN design (evolved modules and blueprints), which benefited training to convergence (section IV), thus resulting in a task-performance boost (compared to base-CDN). Specifically, MMS-CDN achieves a maximum accuracy of 91.1% after training to convergence, whereas base-CDN achieves 90.1% (Table III).

The improved efficacy of MMS-CDN is also hypothesized to be a result of the MMS-CDN speciation extensions (section III-B). Module speciation enabled the evolution of functionally specialized modules within blueprints (encoding CNN designs), which in turn enabled improvements in overall CNN task-performance. This is supported by results in previous work [24], [25]. However, in this study our *centroid representative selection* in speciation ensured that functionally incongruous outlier modules would not be selected for and thus not inhibit overall behavioural efficacy of evolving blueprints.

The task-performance boost enabled by elitism and speciation extensions is reflected in the average accuracy over generations ($green$ and red solid lines in Figure 4 for base-CDN versus MMS-CDN, respectively), indicating the blueprint populations (over all 50 generations) of MMS-CDN yield an approximately 20% higher accuracy compared to base-CDN.

However, the maximum accuracy of all method variants ($green$, red and $blue$ solid lines in Figure 4 for base-CDN, MMS-CDN, and MMS-DA-CDN, respectively), increases slowly between generations 10 and 50, indicating that making significant improvements in blueprint task-performance becomes increasingly difficult at higher accuracy scores. This is supported by related work that similarly evolved CNN architectures and demonstrated only marginal task-performance gains above given classification accuracy thresholds [30], [31].

Second, Mann–Whitney U tests compared the accuracy of MMS-CDN and MMS-DA-CDN evolved and trained CNNs. Results indicate MMS-CDN average maximum and average (Figure 4) accuracy as significantly higher ($p < 0.01$). However, MMS-DA-CDN achieves the highest maximum and average accuracy after being trained to convergence (Table III). During neuro-evolution, MMS-DA-CDN yields maximum and average accuracy significantly higher ($p < 0.01$) than base-CDN ($blue$ and $green$ lines in Figure 4).

The demonstrated higher accuracy of CNNs trained to convergence after neuro-evolution by MMS-DA-CDN is hypothesized to be the result of improved data-augmentation[5] evolved by MMS-DA-CDN. However, given the lack of related research on evolving data-augmentation [32], the exact computational mechanisms responsible for the higher accuracy of MMS-DA-CDN remains the topic of ongoing research.

Increasing the number of epochs used in evolution to 25 (as in *AmoebaNet* [21]) yielded significant convergence accuracy increases over our results measured with 8 epochs in evolution. This can be seen in Table III where the maximum accuracy of the MMS-DA-CDN (25 epoch variant) is 94.5% compared to the MMS-DA-CDN (8 epoch variant), which yielded a maximum accuracy of 92.8%.

Overall, these results support hypotheses from related work [33] demonstrating benefits of suitable neuro-evolution elitism (boosting exploitation in evolutionary search [34]), and speciation operators (diversifying the population and encouraging function specialisation [23]). Hence, this study's results support our main research objective (section I), via demonstrating the evolutionary benefits of these elitism and speciation extensions (MMS-CDN in Figure 4), and further accuracy benefits to CNNs of evolutionary data-augmentation and post-evolution training (MMS-DA-CDN in Table IV).

[5]An overview of the evolved DA schemes used by the highest performing blueprints is available at: https://github.com/sash-a/CoDeepNEAT/

To thoroughly test the efficacy of our CDN method variants, ongoing research is applying these variants beyond CIFAR-10, to a broad range of publicly available computer-vision datasets. Current research is also running experiments to further investigate the exact contributions of the elitism, speciation and evolutionary data-augmentation extensions to the overall accuracy of evolved and trained CNNs, especially in comparison to established $AutoML$ methods such as *AmoebaNet* [21]. Such experiments are also testing extended neuro-evolution run-times and training on broad range of data-sets.

VI. Conclusion

This paper presented experiments demonstrating novel $elitism$, $speciation$ and evolutionary $data\ augmentation$ operators applied as extensions to the CoDeepNEAT neuro-evolution based deep-learning method. This benefited the neuro-evolution and automated design of *Convolutional Neural Networks*, subsequently trained on a popular computer-vision data-set (CIFAR-10). These extensions constituted CoDeepNEAT variants, then compared to a re-implementation of the original CoDeepNEAT method. Results indicated significant task-performance (accuracy classification) improvements for all method variants (extensions) when compared to our CoDeepNEAT re-implementation.

References

[1] Y. Le Cun, Y. Bengio, and G. Hinton, "Deep Learning," *Nature*, vol. 521, pp. 436–444, 2015.

[2] R. Cichy, A. Khosla, D. Pantazis, A. Torralba, and A. Oliva, "Comparison of Deep Neural Networks to Spatio-Temporal Cortical Dynamics of Human Visual Object Recognition Reveals Hierarchical Correspondence," *Scientific Reports*, vol. 6, pp. 1–13, 2016.

[3] G. Hinton and etal, "Deep Neural Networks for Acoustic Modeling in Speech Recognition," *IEEE Signal Processing Magazine*, vol. 29, pp. 82—-97, 2012.

[4] D. Silver and etal, "Mastering the Game of Go with Deep Neural Networks and Tree Search," *Nature*, vol. 529, pp. 484–489, 2016.

[5] J. Vamathevan and et al, "Applications of Machine Learning in Drug Discovery and Development," *Nature Reviews Drug Discovery*, vol. 18, p. 463–477, 2019.

[6] Y. LeCun and etal, "Handwritten Digit Recognition with a Back-Propagation Network," *Proc. Advances in Neural Information Processing Systems*, pp. 396–404, 1990.

[7] Y. Le Cun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based Learning Applied to Document Recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[8] H. Touvron, A. Vedaldi, M. Douze, and H. Jegou, "Fixing the Train-Test Resolution Discrepancy," in *Advances in Neural Information Processing Systems*, H. Wallach and etal., Eds., 2019, pp. 8252–8262.

[9] Z. Hussain, F. Gimenez, D. Yi, and D. Rubin, "Differential Data Augmentation Techniques for Medical Imaging Classification Tasks," in *AMIA Annual Symposium Proceedings*, vol. 2017, 2017, p. 979.

[10] L. Taylor and G. Nitschke, "Improving Deep Learning with Generic Data Augmentation," in *Proceedings of the IEEE Symposium Series on Computational Intelligence*, 2018, pp. 1542–1547.

[11] D. M. Hawkins, "The Problem of Overfitting," *Journal of Chemical Information and Computer Sciences*, vol. 44, no. 1, pp. 1–12, 2004.

[12] E. Cubuk, B. Zoph, D. Mané, V. Vasudevan, and Q. Le, "AutoAugment: Learning Augmentation Strategies From Data," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 113–123.

[13] J. Lemley, S. Bazrafkan, and P. Corcoran, "Smart Augmentation Learning an Optimal Data Augmentation Strategy," *IEEE Access*, vol. 5, pp. 5858–5869, 2017.

[14] A. Zahangir and etal, "A State-of-the-Art Survey on Deep Learning Theory and Architectures," *Electronics*, vol. 292, no. 1, pp. 1–67, 2019.

[15] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated Machine Learning: Methods, Systems, Challenges*. Berlin, Germany: Springer, 2019.

[16] E. Real, A. Aggarwal, Y. Huang, and Q. Le, "Regularized Evolution for Image Classifier Architecture Search," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019, pp. 4780–4789.

[17] R. Olson and J. Moore, "Tpot: A Tree-based Pipeline Optimization Tool for Automating Machine Learning," in *Proceedings of the Workshop on Automatic Machine Learning*, 2016, pp. 66–74.

[18] H. Yu, Q. Han, J. Li, J. Shi, G. Cheng, and B. Fan, "Search What You Want: Barrier Panelty NAS for Mixed Precision Quantization," in *Proceedings of the 16th European Conference on Computer Vision*, 2020, pp. 1–16.

[19] M. Feurer and F. Hutter, "Hyperparameter Optimization," in *Automated Machine Learning*. Springer, Berlin, Germany, 2019, pp. 3–33.

[20] A. Eiben and J. Smith, *Introduction to Evolutionary Computing (2nd edition)*. Berlin, Germany: Springer, 2015.

[21] E. Real, A. Aggarwal, Y. Huang, and Q. Le, "Regularized Evolution for Image Classifier Architecture Search," in *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence*, 2019, pp. 1–10.

[22] R. Miikkulainen, "Neuroevolution," in *Encyclopedia of Machine Learning*, C. Sammut and G. Webb, Eds. Springer, 2010, pp. 716–720.

[23] K. Stanley and R. Miikkulainen, "Evolving Neural Networks through Augmenting Topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[24] R. Miikkulainen and et al, "Evolving Deep Neural Networks," in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, R. Kozma and etal., Eds. Elsevier, 2019, pp. 293–312.

[25] J. Liang and et al., "Evolutionary Neural AutoML for Deep Learning," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019, pp. 401–409.

[26] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6(2), pp. 182–197, 2002.

[27] O. Sporns, D. Chialvo, M. Kaiser, and C. Hilgetag, "Organization, Development and Function of Complex Brain Networks," *Trends Cogn. Sci.*, vol. 8, pp. 418–425, 2004.

[28] E. Real, A. Aggarwal, Y. Huang, and Q. Le, "Regularized Evolution for Image Classifier Architecture Search," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018, pp. 4780–4789.

[29] B. Flannery, S. Teukolsky, and W. Vetterling, *Numerical Recipes*. Cambridge, UK: Cambridge University Press, 1986.

[30] B. V. T. Sinha and A. Haidar, "Optimization of convolutional neural network parameters for image classification," in *Proceedings of the IEEE Symposium Series on Computational Intelligence*, 2017, pp. 1–7.

[31] T. Elsken, J. H. Metzen, and F. Hutter, "Neural Architecture Search: A Survey," *Journal of Machine Learning Research*, vol. 20, pp. 1–21, 2019.

[32] J. Correia, T. Martins, and P. Machado, "Evolutionary Data Augmentation in Deep Face Detection," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019, pp. 163–164.

[33] K. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Machine Intelligence*, vol. 1, pp. 24–35, 2019.

[34] X. Cui, W. Zhang, Z. Tuske, and M. Picheny, "Evolutionary stochastic gradient descent for optimization of deep neural networks," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2018, pp. 6051–6061.