Effective Mutation and Recombination for Evolving Convolutional Networks

Binay Dahal[1] and Justin Zhan[2]

Abstract— **Major part of the success that we have had in deep learning is attributed to the proper design of the architecture of the network model. With all the hardware resources and mathematical foundation at hand, it is the clever assembling of the pieces that defines the performance of that deep learning model. Almost all of the research in this field till date use the models that are designed by human researcher using their past experiences or applying the iterative process of adding components to the model to see which one performs the best. As we strive towards the general Artificial Intelligence, this process of manually tuning the network to make it work on a specific task does not add up well for the cause. We are concerned with automating the task of network architecture design where the learning algorithm tries to discover the best performing model on itself. To do this, we employ evolving algorithm in addition to the gradient descent to evolve and train the model. Specifically, we use a form of genetic algorithm with mutation and recombination operators to constantly change the architecture, while the commonly used gradient descent is used as a learning algorithm for each of the genetically procreated models. We propose series of mutation operators and a method of recombination called Highest Varying k-Features Recombination(HVk-FR) to evolve the CNN models. Results show using our method of recombination on top of mutation yields the best accuracy.**

I. INTRODUCTION

Deep learning and especially convolutional nets have achieved state of the art results over the years in various natural language related and vision related tasks. As much as these success is down to the better learning mechanisms and powerful hardware resources, better design of the network architecture has equally played its part. It is only after the breakthrough in 2012 in Image Classification task by AlexNet[1], the revolution in deep learning and computer vision took off. It reduced the top 5 test error rate to around 16% which was quite a remarkable improvement over other models which had error rate of more than 25%. The breakthrough in the result reported by AlexNet is not due to some novel mathematical innovation. Rather, it combined the pieces of network units, layers and learning methods that were already there and designed a effective network architecture. In the later years, various other models emerged with even better test error rate for the ImageNet dataset. [2],[3],[4],[5] all proposed a better network design and subsequently reduced the test error rate way lower than that of [1]. [5] reported that it's test error rate on image classification is 3.57%. All these works go in a direction of searching a learning model configuration that can represent the data better by either increasing the depth of the model, varying the filter size, or by proposing a novel way of visualizing feature maps(deconvolution)[2], or by proposing a model which comprises of series of sub modules called inception module[4]. All these works are essentially performing the search of optimum network model in the huge search space of possible network architecture. By network architecture we mean all the hyper-parameters including the layer type, depth of the model, filter size(if conv nets are used), learning algorithm, activation function etc. that can be tuned. Hence, the outcome of this search is determined by the experience and expertise of human researcher who is empirically trying out various network layouts. It would make more sense if we can automate this process of network discovery without the manual aid of human researchers. Using some sensible operators, the algorithm explores this high dimensional hyper-parameters space. This is one of the motivation of attempting to evolve the deep models starting with the primitive configuration and gradually discovering the models that best represent the task at hand.

Through this research, we attempt to automate the process of network design by discovery of the hyperparameters that can be tuned in the network. Doing this, we also make progress on the promise of neuroevolution. A subfield with Artificial Intelligence(AI) and Machine Learning (ML), Neuroevolution is concerned with simulating an evolutionary process that resulted our brains, but inside a computer. In other words, neuroevolution seeks to develop the means of evolving neural networks through evolutionary algorithms. In our case, the evolutionary algorithm we use is a genetic algorithm where mutation and recombination operators work together to evolve the neural networks. Our contribution through this paper is three fold: First we present a simple way of encoding neural networks. Each layer of a network is encoded as a node in the list. As new layers are added during the course of evolution, a new node is appended to the list. The node contains all the information specific to a neural network such a type of layer, type of activation function used, size of filter, stride length etc. (in case the layer is convolutional). Then we propose some mutation operators that are applied to the neural networks. Although, there are research that have used mutation to evolve CNN, our objective is to add recombination on top of the mutation operators to improve the result. Hence, we give our own set of simple mutation operators. Lastly, we propose our own recombination operators that is applied between two parent CNN in hope of producing a improved child.

[1]B. Dahal is a Ph.D student at Department of Computer Science, University of Nevada, Las Vegas binay.dahal@unlv.edu

[2]J. Zhan is a Professor at Department of Computer Science, University of Nevada, Las Vegas justin.zhan@unlv.edu

Section II talks about some of the related works in the literature. We will explain briefly the research that have been done in the field of neuroevolution and then give a brief account specifically of evolving deep neural networks using some form of genetic operators. In Section III, we describe our approach and algorithms. In Section IV, we describe the experimental setup and results. We discuss our results in Discussion section and we conclude our paper with conclusion in section VI.

II. Related Works

There are numerous research done in the field of neuroevolution. The commencement of research in neuroevolution happened at around 1980s. The first algorithm in the field was developed as an alternative to the conventional backpropagation algorithm which was used to train the ANN. In the earlier systems, the researchers from neuroevolution community decided themselves on the architecture of the model: which neurons is connected to which one and just use the evolution to learn the weights of the networks. The evolution process just replaced the Stochastic Gradient Descent process. This approach came to be known as fixed-topology neuroevolution, since the architecture remains same throughout the evolutionary process. This fixed-topology neuroevolution had a problem in that the genes of evolving ANNs literally encode their weights. That means, they are "born" knowing everything they will ever know and cannot learn anything further during their "lifetime". The models generated by fixed-topology neuroevolution never gets bigger whereas our brain has evolved into the complex structure from the relatived simpler structure. As a result, we started to see the experiments that tweak both the topology and weight through evolution. This is called topology and weight evolving ANNs (TWEANNs). Using this new flexible approach ,evolution can change the architecture (topology) of a parent ANN slightly in its offspring, such as by adding a new connection or a new neuron.

There were few problems while trying to evolve network architecture using some form of evolutionary algorithms. One of them is called "Competing Convention" problem. This problem means it is hard to combine two parent ANNs(i.e. to apply recombination) to produce an offspring because the different architectural representation of two parents might represent the same functionality. There is no way to distinguish their functionality based on their architecture. Hence, if we apply recombination between the parents with different architecture but same functionality, we might lose some valuable information. Another problem is the tendency for new architectures to go extinct from the evolving population just because they have lower fitness value than other but could have performed better later.

By trying to overcome such problems, Stanley et al[6], proposed a method of evolving network topologies called NeuroEvolution of Augmenting Topologies(NEAT). They claim their method outperformed the best fixed-topology method on a challenging benchmark reinforcement learning task at that time. The increased efficiency is due to (1) employing a principled method of recombination of different topologies, (2) protecting structural innovation using speciation, and (3) incrementally growing from minimal structure. They perform a series of studies to demonstrate that each component they propose is necessary for the functioning of the system as a whole. At that time NEAT obtained state of the art result in pole-balancing task. Despite the fact that NEAT obtained outstanding result, it employed direct encoding as a way to encode all the connections in it's model. This means, every connection has to be described by a separate gene in the genome. This is feasible if the network has the connection in the order of hundreds, it is inapplicable where the network connections are in billions or even millions order. This problem paved the way for indirect encoding, where the number of genes can be much fewer than the number of connections. One of the most popular indirect encoding that was developed is called "Compositional pattern producing networks"(CPNN)[7]. This research was motivated by the fact that natural DNA can encode complexity on an enormous scale. Researchers tried to achieve the same represantational efficiency in computers by implementing "developmental encodings" [8, 9, 10]. Developmental encodings are concerned with mapping the genotype to the phenotype through a process of growth from a small starting point to a mature form. CPNN use this approach. The mapping to the phenotype is performed without local interaction. This means, independent determination of each individual component of phenotype from every other component. CPNNs through interactive evolution of two-dimensional images show that such an encoding can nevertheless produce structural motifs often attributed to more conventional developmental abstractions. Using the CPNN encoding, [11] proposed a new method called Hypercube-Based Indirect Encoding for Evolving Large-Scale Neural Networks(HyperNEAT). Applying the CPNN encoding, connectivity patterns with symmetries and repeating motifs are produced by HyperNeat by interpreting spatial patterns generated within a hypercube as connectivity patterns in a lower-dimensional space. The advantage to this approach stems from the fact that connective CPNNs can represent the same connectivity pattern at any resolution, allowing ANNs to scale to new numbers of inputs and outputs without further evolution. HyperNEAT is successfully applied through visual discrimination networks containing over eight million connections, to various problems like visual discrimination and food gathering tasks. It concludes by saying that the ability to explore the space of regular connectivity patterns opens up a new class of complex high-dimensional tasks to neuroevolution.

With the success of deep learning due to advancement in hardware resources, attention has been shifted to use these powerful hardware to evolve deep neural networks. One of the such research is done by Google Brain. In their research, "Large-Scale Evolution of Image Classifiers"[12], they try to evolve convolutional neural networks in hope of achieving comparable results in image classification task. They evolve the CNN starting with the simple structure(a 1-layer deep CNN) with the help of several mutation operators

like inserting a convolutional layer, removing a convolutional layer, altering the learning rate, resetting the weights, altering the filter size, etc. They have shown that, given significant computational resources, it is possible to evolve models with accuracy within the range of state of the art result. The accuracy they have obtained for image classification task in CIFAR-10 and CIFAR-100 datasets is 94.6% and 77.0% respectively. They also tried to apply recombination operators in addition to the mutations but achieved no further improvements. This has motivated us to explore the formulation of recombination operators that can be applied together with mutation to achieve higher accuracy.

Another research that is concerned with evolving deep neural networks is called CoDeepNEAT[13]. It extends the existing neuroevolution methods to topology, components, and hyperparameters, such as NEAT to achieve results comparable to best human designs in common and standard tasks of object recognition and language modeling. Another research that applies evolution strategies for reinforcement learning is done in [14]. They explore the use of Evolution Strategies(ES), as an alternative to popular MDP-based RL techniques as as Q-learning and Policy Gradients. Experiments performed on MuJoCo and Atari has demonstrated that Evolutionary Strategy can be used effectively with good scalability.

There are some other research that have applied mutation for evolving networks in reinforcement learning. [15] has developed a family of safe mutation (SM) operators. Safe mutation operators should be able to change itself in a degree that does not vary the behavior of the network too much but should help positively for further exploration. The concern is, while the random mutation works well generally in lower dimensions, a stochastic change in millions of weights might probably break the existing functionality. They compute the degree of mutation to individual weights by calculating the degree of sensitivity of network's output with respect to the weight in contrast to network's cost. Similarly, [16] uses the genetic algorithm to train deep neural network for reinforcement learning to with over 4 millions free parameters. It shows that this gradient free, simple approach can be an alternative to our conventional gradient based training with very good performance of hard RL problems including Atari and humanoid locomotion. With all these research employing some sorts of evolution strategies in place of commonly used stochastic gradient descent. The comparable results between these two approaches is somewhat surprising given the ambiguity on how ES actually relates to SGD. [17] explores on this area of concern and develops SGD based proxy that accurately predicts the performance of different ES population size. [18] shows that Evolution strategies that have performed well in challenging deep reinforcement learning is more than just a traditional finite-difference approximator. [19] performs research on improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents.

There are few attempts at evolving unsupervised deep neural networks compared to supervised deep learning. One of such attempts is done in [20]. It evolves unsupervised networks by incorporating local search strategy to improve the performance. It achieves an error classification rate of 1.15% on MNIST, which is a promising result against state-of-the-art unsupervised DL algorithms. [21, 22, 23] and [24] are the other works of neuroevolution on reinforcement learning and LSTM structure respectively. One related work that use CNN is [25].

III. Proposed Approach

We employ a sort of genetic algorithm in combination with the common gradient descent algorithm used to train a neural network. Genetic algorithm is used to continuously evolve the network architecture while the evolved networks go through gradient descent to learn their parameters. In this work, we are concerned with evolving convolutional networks and apply it to some image classification tasks.

A. Main Idea & the Algorithm

We start with a certain sized population of network models. To let the algorithm discover the appropriate network structure on itself, we start with very primitive network.Each network model in a population is called an individual. The initial individual comprises of a single convolutional layer and an output layer depending on the type of output we require. For example, if we are dealing with image classification tasks with 10 output classes, output is a 10 unit softmax layer. We apply tournament selection method to select individuals that undergo evolution. The main idea is to select 3 individuals from a population and choose the best 2 models based on their fitness as parents. The third one is immediately killed. The parents undergo recombination to produce a child. The newly created child goes through mutation which gives it novel architecture. It is then trained with gradient descent for certain steps. After the training, the child is put into the population and is ready to act as a parent. The idea can be put into algorithmic way as shown in Algorithm 1:

Algorithm 1 Algorithm for evolving convolutional nets to discover appropriate network structure

1: Initialize a population of 50 individuals
2: **while** individuals are still evolving with better fitness **do**
3: Choose 3 individuals randomly
4: Select best 2 individual based on their fitness as parents and kill the third one
5: Apply recombination operator to the parents and create a child as a result of recombination
6: Apply mutation operator on the child
7: Train the newly created child using gradient descent
8: Make the child active by inserting it into the population

B. Encoding of Conv network model

In order to apply the evolutionary operators to the models, they are encoded in a layer wise fashion. Each layer is represented as a node in a list.

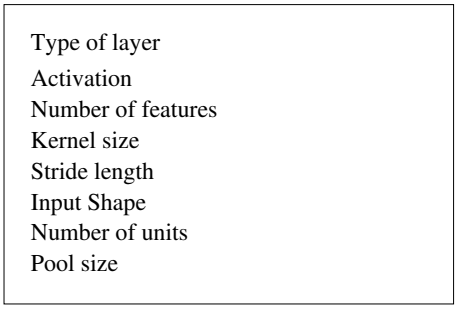


Fig. 1: Encoding of a network layer

- Type of layer: Can be one of convolutional layer, dense layer, or pooling layer
- Activation: Type of activation function used in that layer
- Number of features: Applicable only if the layer is convolutional.
- Kernel size: Applicable only if the layer is convolutional. The size of filter(kernel) to be used
- Stride length: Applicable only if the layer is convolutional. Input shape: The shape of input to be used.
- Number of units: Applicable only if the layer is Dense. The number of neurons to be used in the layer.
- Pool size: Applicable only if it's the pooling layer.

C. Recombination

Integrating recombination with mutation effectively is a challenging task. Above we talked about, how Competing Conventions problem poses a challenge in identifying a functional unit of a neural networks that can undergo recombination. In our work, instead of identifying a unit of general networks, we are concerned with Convolutional Neural Networks only. We propose "Highest Varying k-Features Recombination (HVk-FR)" method as a way to recombine the features between parent models.

1) Highest Varying k-Features Recombination(HVk-FR): In our proposed method (HVk-FR), we identify the convolution layer filters as a functional unit of a network that undergo recombination between parents. The main idea is as follows: Between the parents, the model with the highest fitness, say P_1 combines with the lower scoring parent P_2 along it's convolution filters. The random layer of P_1 inherits 'k' most divergent features from the same random layer of P_2.

Let P_1 has L convolution filters and P_2 has M convolution filters at randomly selected i^{th} layer.

To find the most divergent features of P_2, we compare the features of P_2 with the representative feature of P_1 instead of comparing with each of the features. We use Principal Component Analysis (PCA) to find the feature that best represents the i^{th} layer features of P_1 in lower dimension. We call it $P_{1_{rep}}$.

$P_{1_{fe}} = \{f_1, f_2...f_L\}_L$ is a L dimensional vector of features.

$$P_{1_{rep}} = PCA(P_{1_{fe}})$$

We obtain 3-dimensional representative feature $P_{1_{rep}}$. We now project each of the L features of P_1 on $P_{1_{rep}}$ to obtain L dimensional vector. Each element of this vector is a 3-dimensional projected vector. We call this vector $P_{1_{mapped}}$.

$$P_{1_{mapped}} = P_{1_{fe}}.P_{1_{rep}}$$

Here, . operator denotes projection operation.

$$P_{1_{mapped}} = \{f_{1_m}, f_{2_m}, ..., f_{L_m}\}_L$$

$f_{x_m}; x \in L$ is a 3 dimensional vector.
Similarly, we project features of P_2:

$$P_{2_{fe}} = \{g_1, g_2...g_M\}_M$$

over $P_{1_{rep}}$ to obtain

$$P_{2_{mapped}} = P_{2_{fe}}.P_{1_{rep}}$$

$$P_{2_{mapped}} = \{g_{1_m}, g_{2_m}, ..., g_{M_m}\}_M$$

$g_{x_m}; x \in M$ is a 3 dimensional vector.
We get the projected vectors $P_{1_{mapped}}$ and $P_{2_{mapped}}$. Now, for each projected features of $P_{2_{mapped}}$ we compute the distance with each of the features of $P_{1_{mapped}}$.

$$Dist(g_{x_m}) = \sum_{i=1}^{L} g_{x_m} - f_{i_m}$$

This way we obtain a distance vector $Dist$ where i^{th} element represents the cumulative distance between t^{th} feature of P_2 and all the features of P_1.

$$Dist = \{Dist(g_{1_m}), Dist(g_{2_m}), ..., Dist(g_{M_m})\}_M$$

Now, k-highest value in $Dist$ vector represents the k-features in P_2 that are most divergent with the features of P_1. Finally, we select those k-features and combine with the L features of randomly selected i^{th} layer of P_1. The resulting model after this recombination operation is the model P_1 with $L + k$ features in layer i. This child inherits all the weights of parent P_1 except in the recombined layer i. The motivation behind HVk-FR method is that even the highest performing model can learn something in terms of features from the other lower performing models.

D. Mutation

After a child is created as a result of recombination, it goes through mutation to evolve it's structure. Following are the mutation operators the child can undergo:

- Add_Convolution: Adds a new convolution layer to the existing network.
- Change_filter_size: Changes the size of filter in a convolution layer. The filter size increases or decreases symmetrically on rows and columns.
- Change_Stride_length: Changes the stride length while doing the convolution.
- Identity: Keeps the network as it it.
- Add_Dense_layer: Adds a new Dense layer to the existing network.
- Change_Activation: Changes the activation function among Relu, Sigmoid, etc.

E. Training using the gradient descent

After a child goes through all the evolutionary operators it is ready to be trained. We train the child using the normal gradient descent to certain steps based on the target to be learned. After it is complete, it is put back in the population.

IV. Experimental Setup and Result

A. Initial Encoded Population

We start off with creating an initial pool of population. As we talked before, each individual is encoded as a list of nodes where each node represents a layer on the model. Initially, we have a pool of encoded models. To train these population, we decode the encoded models to the actual models and pass it to the training module. Our experiment starts with a pool of 50 encoded individuals. The base model to start the evolution is a 1 layer deep convolutional neural network. The default number of filters is 4 and the default filter size is (3,3). We start with the stride length of 1. The activation function for the hidden layer of base model is RELU and for the output layer is Softmax. Initial encoded individual is depicted in the following figure:

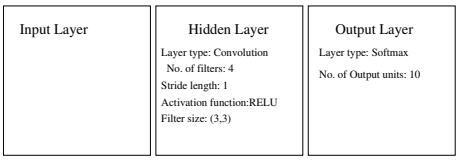


Fig. 2: Initial encoded model to start the training where each box represents a layer

After some generations, as a result of mutation, one of the probable network model that can be seen in a population is shown in figure 3.

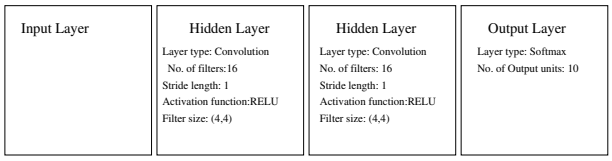


Fig. 3: A probable network model that can be resulted from a series of mutation operators

We only allow safe mutation to take place during the course of evolution. For example, we start with 1 layer deep CNN, so we mustn't allow our algorithm to remove the only convolution layer we started with. Or, we start with a (4,4) filter size. We don't allow the filter size to grow more than the image size. Similar is the case with stride length. Below we show an example of evolved network model that should not be created.

B. Dataset

We ran our experiment for image classification task using CIFAR-10 dataset. It contains 60,000 32*32 color images in 10 classes with 6,000 images per class. There are 50,000 training images and 10,000 test images. Given the small size of the images, it is relatively easier and faster to train on this dataset.

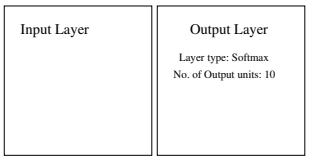


Fig. 4: A network model with no convolutional layer that our algorithm should not create

TABLE I: Average accuracy:Average accuracy of all the network models that are undergoing evolution at the end of generation 10. Best accuracy:Best accuracy among all the models at the end of generation 10.

Type	Mut only	Mut+Rec(1f)	Mut+rec(2f)
Avg. Accuracy	0.474	0.462	0.491
Best Accuracy	0.56	0.55	0.583

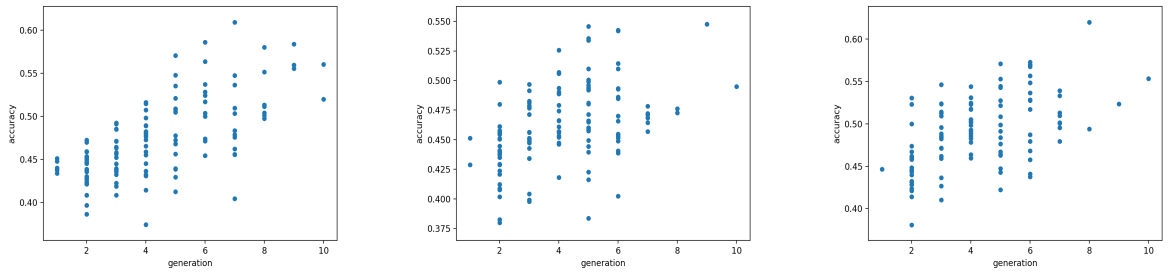
C. Results

We performed various experiments with CIFAR-10 data. First, we ran our experiments using only the mutation operators to evolve the network. Then, we evolved the network models using recombination over mutation. Figure 5 shows the general trend of evolution of models in terms of accuracy over the increasing generation. We conducted our experiments up to 10 generation of evolution. This, by no means produces the state of the art result, however, can work as a proof of concept of our proposed method. We can see in the figure, the accuracy of models generally increases as the models are evolved towards higher generation. All three variants of our experiments (mutation only, mutation+recombination using 1 filter, mutation+recombination using 2 filters) display a similar trend.

Table 1 compares the average and best accuracy of three different types of experiment. When we use Recombination with 1 features with Mutation there is no gain in either of the average or best accuracy of the models whereas recombining 2 features with mutation yields better result than evolving the network using mutation only with an accuracy of 0.58.

V. Discussion

Results from our experiments give a strong indication that if given enough time to evolve the networks, we can find the appropriate network structure with very good accuracy. Our experiments which was run up to 10 generation of evolution is a proof of concept for our proposed method. We will continue our work to let the networks evolve itself to the state of the art level. In figure 5, (a),(b) and (c) shows that in general the accuracy of models increase with generation. In (a) and (c) we can see the highest performing models have accuracy of around 0.60 however from Table 1, the best performing model at the end of generation 10 has accuracy of 0.56 and 0.583 for mutation only and 2-filters



(a) Mutation only

(b) Mutation with recombination using 1 filters only

(c) Mutation with recombination using 2 filters only

Fig. 5: General trend of accuracy with generation

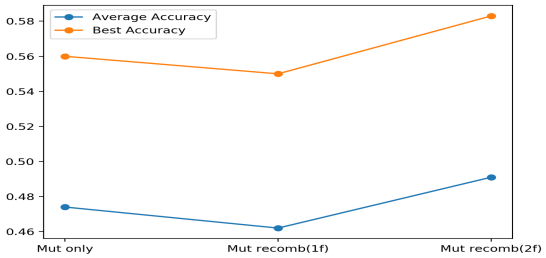
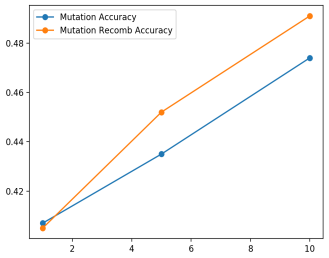
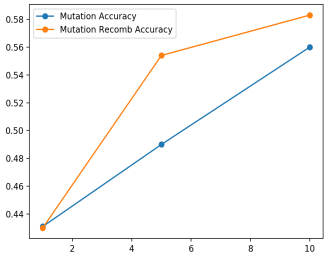


Fig. 6: Comparisons of average and best accuracy among three experiments



(a) Growth of average accuracy among models in generation 1,5 and 10



(b) Growth of best accuracy among models in generation 1,5 and 10

Fig. 7: The accuracy of models monotonously increase when recombination is used with 2 filters and when only mutation is used. However, using recombination always yields higher accuracy than just using mutation.

recombination with mutation, respectively. That is because, some intermediate generation may produce a network model with higher accuracy but as they continue to evolve, the models can evolve into bad structure, hence yielding lower accuracy.

Figure 6 compares the average and best accuracy of models across three variation of experiments we conducted. In both average and best case, recombination using 2 filters yield better accuracy than all others. This verifies that using our method of HVk-FR for recombination on top of mutation is helpful for efficiently evolving the models.

We have evolved our network models for just 10 generation and obtained the better accuracy with the combination of mutation and recombination with 2 filters, we claim, as we go on evolving the models for further generation, the best result is yielded by using mutation and k-filters recombination method that we proposed. Figure 7 supports our claim. (a) and (b) shows the average and best accuracy of model. In generation 1, we start our experiments with almost same accuracy of models. When we reach generation 5, 2-filters recombination with mutation takes a lead over mutation only and the trend continues till generation 10. Hence, we can extrapolate this result and claim that the state of the art result for CIFAR-10 dataset will be obtained by further evolving networks using k-filters recombination with mutation. We will keep on training the models for further generation and verify our claim. Using further more filters for recombination may produce even better result, however, that test hasn't been done. Using 1-filter HVk-FR didn't improve the result but 2-filters did. So, we can speculate that more than 2-filter recombination may produce a higher accuracy.

VI. Conclusion

We explored the area of neuroevolution where instead of hard coding a network architecture, an optimum hyper-parameters of the network are searched using some form of evolution strategies. In particular, we employed genetic algorithm developing a series of mutation and recombination operators and used them to evolve the network structure. To do this, we established a network encoding where each nodes represent a network layer. Phenotype of the networks

can be easily obtained from values of a genotype. We then proposed a series of mutation operators to act on a network model to improve it. Our major contribution lies in proposing Highest Varying k-Features Recombination(HVk-FR) method for recombination. To the best of our knowledge, there are no other works that have reported the positive application of mutation and recombination together while evolving CNNs.

ACKNOWLEDGMENT

To be added later

REFERENCES

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[2] Matthew D Zeiler and Rob Fergus. "Visualizing and understanding convolutional networks". In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

[3] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[4] Christian Szegedy et al. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[5] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[6] Kenneth O Stanley and Risto Miikkulainen. "Evolving neural networks through augmenting topologies". In: *Evolutionary computation* 10.2 (2002), pp. 99–127.

[7] Kenneth O Stanley. "Compositional pattern producing networks: A novel abstraction of development". In: *Genetic programming and evolvable machines* 8.2 (2007), pp. 131–162.

[8] Petet J Bentley and SP Kumar. "The ways to grow designs: A comparison of embryogenies for an evolutionary design problem". In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 99)*, pp. 35–43.

[9] Gregory S Hornby and Jordan B Pollack. "Creating high-level components with a generative representation for body-brain evolution". In: *Artificial life* 8.3 (2002), pp. 223–246.

[10] Kenneth O Stanley and Risto Miikkulainen. "A taxonomy for artificial embryogeny". In: *Artificial Life* 9.2 (2003), pp. 93–130.

[11] Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. "A hypercube-based encoding for evolving large-scale neural networks". In: *Artificial life* 15.2 (2009), pp. 185–212.

[12] Esteban Real et al. "Large-scale evolution of image classifiers". In: *arXiv preprint arXiv:1703.01041* (2017).

[13] Risto Miikkulainen et al. "Evolving deep neural networks". In: *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Elsevier, 2019, pp. 293–312.

[14] Tim Salimans et al. "Evolution strategies as a scalable alternative to reinforcement learning". In: *arXiv preprint arXiv:1703.03864* (2017).

[15] Joel Lehman et al. "Safe mutations for deep and recurrent neural networks through output gradients". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2018, pp. 117–124.

[16] Felipe Petroski Such et al. "Deep neuroevolution: genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning". In: *arXiv preprint arXiv:1712.06567* (2017).

[17] Xingwen Zhang, Jeff Clune, and Kenneth O Stanley. "On the Relationship Between the OpenAI Evolution Strategy and Stochastic Gradient Descent". In: *arXiv preprint arXiv:1712.06564* (2017).

[18] Joel Lehman et al. "ES is more than just a traditional finite-difference approximator". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. 2018, pp. 450–457.

[19] Edoardo Conti et al. "Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents". In: *Advances in Neural Information Processing Systems*. 2018, pp. 5027–5038.

[20] Yanan Sun, Gary G Yen, and Zhang Yi. "Evolving Unsupervised Deep Neural Networks for Learning Meaningful Representations". In: *IEEE Transactions on Evolutionary Computation* (2018).

[21] Justin Bayer et al. "Evolving memory cell structures for sequence learning". In: *International Conference on Artificial Neural Networks*. Springer. 2009, pp. 755–764.

[22] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. "An empirical exploration of recurrent network architectures". In: *International Conference on Machine Learning*. 2015, pp. 2342–2350.

[23] Bowen Baker et al. "Designing neural network architectures using reinforcement learning". In: *arXiv preprint arXiv:1611.02167* (2016).

[24] Barret Zoph and Quoc V Le. "Neural architecture search with reinforcement learning". In: *arXiv preprint arXiv:1611.01578* (2016).

[25] Shreyas Saxena and Jakob Verbeek. "Convolutional neural fabrics". In: *Advances in Neural Information Processing Systems*. 2016, pp. 4053–4061.