A First Step Toward Incremental Evolution of Convolutional Neural Networks

Dustin K. Barnes, Sara R. Davis, Emily M. Hand, Sushil J. Louis
University of Nevada, Reno
Reno, Nevada
{dkbarnes,sarad}@nevada.unr.edu,{emhand,sushil}@unr.edu

ABSTRACT

We introduce a novel algorithm – ConvNEAT – that evolves a convolutional neural network (CNN) from a minimal architecture. Convolutional and dense nodes are evolved without restriction to the number of nodes or connections between nodes. The proposed work advances the field with ConvNEAT's ability to evolve arbitrary minimal architectures with multi-dimensional inputs using GPU processing.

CCS CONCEPTS

• **Computing methodologies** → **Genetic algorithms**;

KEYWORDS

Convolutional Neural Networks

ACM Reference Format:
Dustin K. Barnes, Sara R. Davis, Emily M. Hand, Sushil J. Louis. 2020. A First Step Toward Incremental Evolution of Convolutional Neural Networks. In *Proceedings of The Genetic and Evolutionary Computation Conference 2020 (GECCO '20 Companion).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3377929.3389916

1 INTRODUCTION

Convolutional(conv) Neural Networks (CNNs) are widely used in machine learning, from computer vision to natural language processing. Common CNN architectures are modeled after the initial work in [2], and rely on a series of feature maps leading to fully connected layers as shown in Figure 1a. Evolutionary methods have been applied to CNNs, but with significant constraints on the architectures generated [4, 5]. Multi-dimensional inputs pose challenges due to shape dependencies between convolutional layers and dense layers. The intricate relationship between dense and conv layers is what makes evolving a CNN so challenging.

The EXACT network evolves a CNN architecture, avoiding the issue of layer incompatibility by not evolving dense layer architecture, instead using a fully convolutional approach [1]. EXACT represents the conv layers as edges, rather than nodes, and uses a fixed number of conv layers, as well as fixed parameters for max pooling. EvoCNN evolves blocks of convolutions, padding the output to ensure that shapes are compatible [3]. While [3] allows for a broader range of architectures than [1], information is lost each time a layer is padded. Both methods are computationally expensive, taking months [1] to train on CPU and several days [3] to train on GPU.

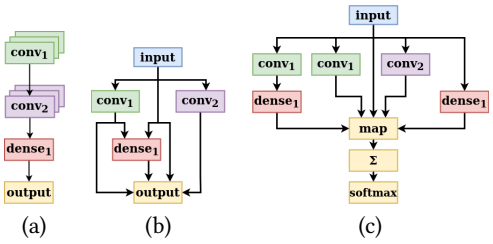

Figure 1: (a) a typical CNN, (b) a genome generated by ConvNEAT, (c) all individual paths in the genome generated by ConvNEAT and the associated outputs.

In this work, we introduce ConvNEAT, a method for evolving CNNs composed of both conv and dense layers removing the aforementioned architecture constraints. ConvNEAT is a dynamic algorithm that evolves a CNN from a minimal architecture allowing for dense connections, any number of conv layers with variable shapes, unconstrained connections between layers, and weights that maximize object classification fitness across ten classes. An example of the evolved architecture can be seen in Figure 1b & c.

The proposed ConvNEAT algorithm is the only method that utilizes layer reshaping to allow for the evolution of both conv and dense layers in a CNN architecture. With this freedom, ConvNEAT can develop unique architectures that human engineers might not consider.

2 PROPOSED METHOD

In this section, we outline the ConvNEAT algorithm for architecture selection. First, an initial population of 100 parents is generated consisting of a single input node directly connected to a single output node with random weight initialization. The parents are copied to create 100 new children, and mutation is performed on the children. Mutations may add additional conv or dense nodes to increase the complexity of the network. Backpropagation is performed for 50 epochs to train model weights after mutation. The 100 fittest individuals from the collection of 100 parents and 100 children are carried into the next generation. ConvNEAT uses categorical cross-entropy as its loss function, which is represented as $\sum_{M}^{c=1} y_{o,c} \log(p_{o,c})$, where M is the number of classes, $y_{o,c}$ is a 1 or a 0 that corresponds to a correct classification of class label (c) or observation (o), and p is a prediction about which class the

GECCO '20 Companion, July 8–12, 2020, Cancun, Mexico

ACM ISBN 978-1-4503-7127-20/07…$15.00
https://doi.org/10.1145/3377929.3389916

observation belongs to. As loss in a neural network is a measurement of error, the fitness function for ConvNEAT maximizes the inverse square of the loss after performing mutation and backpropagation on the children. This process is repeated for 15 generations. The MNIST dataset [2] is used for evaluation for comparison with previous works [1, 3].

Each genome consists of a set of nodes (input, output, conv and/or dense) and edges (as seen in Figure 1). Output from one node is passed as input to the next node. Nodes can either perform convolutions (conv) or flattening (dense) on their input. **Input and output nodes** represent the desired input and output shapes of the network respectively. The output node stores additional information about the viable paths through the network. **Conv nodes** take input of a size $c \times k$ (output size of previous node) and perform a single $n \times m$ conv on it, where n and m are randomly chosen during evolution. Each conv node stores the weights for its filter, and the same filter is applied to all inputs that pass through that node. Typical neural networks use multiple conv filters per conv layer, whereas our nodes consist of a single filter; this difference can be seen in Figure 1b & c, where none of the conv layers are stacked. When backpropagation occurs, the weights for a node are updated according to the gradients calculated by every path through the node. **Dense nodes** with one or more neurons, take either input shaped by previous conv layers or flattened input from a previous dense layer, and create connections between the previous layer and the output layer or another dense layer. Dense connections, as well as the connections to the output store weights for each path that travels through. This allows inputs of various dimensions to be passed through the same dense node. An example of varying paths passing through a conv layer is shown in Figure 1b & c.

Five primary methods of mutation are implemented, and outlined below:

Add Convolutional Node with a randomly shaped and randomly initialized filter to the network. The generated node is limited such that it may only be added between two shaped nodes (eg, two conv or an input and a conv). Each time a node is added, a new path is generated.

Add Dense Node, connecting a shaped (conv) node to another dense node, or connecting the dense node to an output node.

Add Edge selects a path, and then generates a new path by inserting an existing node into the selected path.

Delete Node selects a node, and then removes the node from the network and from every path containing the node.

Delete Edge selects a path, and then removes the a node from the path. If the node does not exist in any other path, it is removed from the entire genome.

3 RESULTS AND DISCUSSION

We test our method on the MNIST dataset, using a variety of different mutation rates. Nodes are added at either 20% and 80%, for both dense and conv, with edges being added at rates of 0%, 30%, 50%, or 80%. Deletion occurs with a 30, 50 and 80% chance. Notably, despite the variations in mutation rates, the standard deviation in average accuracy between all 375 runs was only 0.004.

On the MNIST dataset, our evolved networks achieve 94.7% accuracy, which is slightly lower than the work in [1, 3]. Three of the networks created by ConvNEAT are shown in Figure 2. The

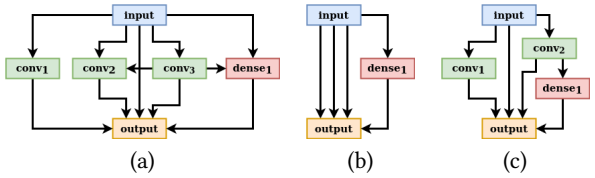

Figure 2: Neural network architectures generated by ConvNEAT and associated accuracies; (a) an eliminated architecture (78.6%), (b) a high fitness individual with simple architecture (92.9%), (c) a high fitness individual with complex architecture (93.8%)

most fit individuals generated by ConvNEAT typically do not have more than 2 conv layers, and only around 25% of the 375 fittest individuals generated contain a dense layer. The generated networks show some of the potential pitfalls of our current implementation - all contain a direct mapping from inputs to outputs, sometimes multiple. While this is viable due to the simplicity of MNIST, the addition of multiple such mappings provides little benefit, and is the result of paths being added and edges being deleted, but being unable to remove the paths entirely during mutation. Interestingly, ConvNEAT does not appear to converge on a single architecture when training on MNIST. Rather, it generates some architectures, like the one in Figure 2b, that closely resemble traditional neural network architectures, like the one in Figure 1a, and others, like Figure 2c, that human engineers are extremely unlikely to create.

It is worth noting that using TensorFlow 2 on a single GTX 1080TI, our work provides a 3.5x speedup over the 2-3 days reported in other studies [3]. We expect that the addition of crossover will not significantly increase the run-time of ConvNEAT, as the computation time is insignificant relative to the time needed to perform weight updates through backpropagation.

In order to expand on the work established here, we intend to implement several methods of crossing over, to avoid getting stuck in local maxima, and approach state of the art accuracy achieved in [1, 3]. In order to make our work more applicable in computer vision applications, we plan to evaluate our method using more complex datasets, such as CIFAR-10, and eventually, CelebA.

ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation under Grant No. 1909707. This work was also supported in part by grant N00014-17-1-2558 from ONR and grant 69A3551747126 from DoT. Standard disclaimers apply.

REFERENCES

[1] T. Desell. 2017. Developing a Volunteer Computing Project to Evolve Convolutional Neural Networks and Their Hyperparameters. In *2017 IEEE 13th International Conference on e-Science (e-Science)*. 19–28. https://doi.org/10.1109/eScience.2017.14

[2] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*. 2278–2324.

[3] Yanan Sun, Bing Xue, and Mengjie Zhang. 2017. Evolving Deep Convolutional Neural Networks for Image Classification. *CoRR* abs/1710.10741 (2017). arXiv:1710.10741 http://arxiv.org/abs/1710.10741

[4] Lingxi Xie and Alan L. Yuille. 2017. Genetic CNN. *CoRR* abs/1703.01513 (2017). arXiv:1703.01513 http://arxiv.org/abs/1703.01513

[5] Barret Zoph and Quoc V. Le. 2016. Neural Architecture Search with Reinforcement Learning. *CoRR* abs/1611.01578 (2016). arXiv:1611.01578 http://arxiv.org/abs/1611.01578