A Genetic Programming Approach to Design Convolutional Neural Network Architectures

1st Autor
Institution
Street Adress
City, Country Post Code
emailadress

2nd Autor
Institution
Street Adress
City, Country Post Code
emailadress

3rd Autor
Institution
Street Adress
City, Country Post Code
emailadress

ABSTRACT

Convolutional neural network (CNN), which is one of the deep learning models, has seen much success in a variety of computer vision tasks. However, designing CNN architectures still requires expert knowledge and much trial and errors. In this paper, we attempt to automatically construct the CNN architectures for an image classification task based on Cartesian genetic programming (CGP). In our method, we adopt the highly-functional modules such as convolutional blocks and tensor concatenation as the node functions in CGP. The CNN structure and connectivity represented by the CGP encoding method are optimized to maximize the validation accuracy. To evaluate the proposed method, we constructed the CNN architecture for the image classification task with the CIFAR-10 dataset. The experimental result shows that the proposed method can automatically find the competitive CNN architecture compared with state-of-the-art models.

KEYWORDS

genetic programming, convolutional neural network, designing neural network architectures, deep learning

ACM Reference format:
1st Autor, 2nd Autor, and 3rd Autor. 2017. A Genetic Programming Approach to Design Convolutional Neural Network Architectures. In *Proceedings of the Genetic and Evolutionary Computation Conference 2017, Berlin, Germany, July 15–19, 2017 (GECCO '17),* 5 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Deep learning, which uses deep neural networks as a model, has shown a good performance on many challenging artificial intelligence and machine learning tasks such as image recognition [17, 18], speech recognition [11], and reinforcement learning tasks [20, 21]. In particular, convolutional neural networks (CNNs) [18] have seen huge success in image recognition tasks in the last few years and is applied to various computer vision application, e.g. GAN [7], colorization [34], image to text anotation [33]. A commonly used CNN architecture mainly consists of several convolutions, pooling, and fully connected layers. Several recent studies focus on developing the novel CNN architecture that achieves higher classification accuracy, e. g., GoogleNet [30], ResNet [10], and DensNet [12]. Despite their success, designing CNN architectures is still a difficult task since there exist many design parameters such as the depth of a network, the type and parameters of each layer, and the connectivity of the layers. The state-of-the-art CNN architectures have become deep and complex, which suggests that a significant number of design parameters should be tuned to realize the best performance for a specific dataset. Therefore, the trial and error or expert knowledge are required when the users construct the suitable architecture for their target dataset. Because of this situation, the automatic design methods for the CNN architectures is highly beneficial.

The neural network architecture design can be viewed as the model selection problem in the machine learning. The straightforward approach is to deal with the architecture design as the hyperparameter optimization problem, optimizing the hyperparameters such as the number of layers and neurons using black-box optimization techniques ++TODO: []. ()++

Evolutionary computation has been traditionally applied to design the neural network architectures ++TODO: ref. X. Yao's review, Kitano, etc... ()++. There are two types of encoding schemes for the network representation: direct and indirect coding. The direct coding represents the number and connectivity of neurons directly as the genotype, whereas the indirect coding represents a

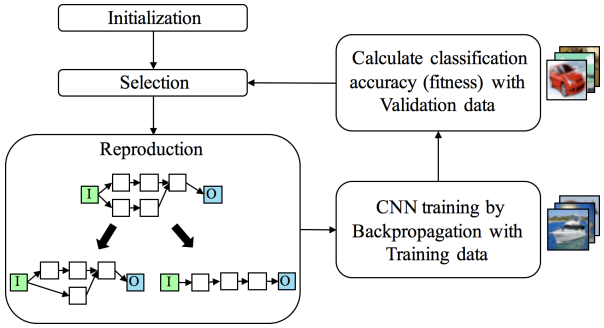

Figure 1: Overview of our method. Our method searches a CNN architecture using genetic programming. That CNN architecture is then trained on a learning task, and returns an accuracy. The network search is performed to maximize the accuracy by the evolutionary algorithm. [Note (Shinichi): I cannot understand above figure...]

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
GECCO '17, Berlin, Germany
© 2017 Copyright held by the owner/author(s). 978-x-xxxx-xxxx-x/YY/MM…$15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

generation rule of the network architectures. While almost traditional approaches optimize the number and connectivity of low-level neurons, the modern neural network architectures for deep learning have many units and various types of units, e.g., convolution, pooling, and normalization. Optimizing those many parameters in reasonable computational time may be difficult. Therefore, the use of the highly-functional modules as a minimum unit is promising.

In this paper, we attempt to design CNN architectures based on a genetic programming. We use the Cartesian genetic programming (CGP) [8, 19] encoding scheme, one of the direct encoding, to represent the CNN structure and connectivity. The advantage of this representation is its flexibility; it can represent variable-length network structure and the skip connections. Moreover, we adopt the relatively highly-functional modules such as convolutional blocks and tensor concatenation as the node functions in CGP to reduce the search space. To evaluate the architecture represented by the CGP, we train the network using training dataset in an ordinary way. Then the performance for another validation dataset is assigned as the evaluation of the architecture. Based on this fitness evaluation, an evolutionary algorithm optimizes the CNN architectures. To check the performance of the proposed approach, we conducted the experiment constructing the CNN architecture for the image classification task with the CIFAR-10 dataset [16]. The experimental result shows that the proposed approach can automatically find the competitive CNN architecture compared with state-of-the-art models.
[Note (Shinichi): It would be nice if we can describe the research question clearly.]

The rest of this paper is organized as follows. The next section presents related work on the neural network architecture design. In Section 3, we describe our genetic programming approach to design the CNN architectures. We test the performance of the proposed approach through the experiment. Finally, in Section 5, we describe our conclusion and future work.

2 RELATED WORK

This section briefly reviews the related work on the automatic neural network architecture design: hyperparameter optimization, evolutionary neural networks, and reinforcement learning approach.

2.1 Hyperparameter Optimization

We can consider the neural network architecture design as the model selection or hyperparameter optimization problem from machine learning perspective. There are many hyperparameter tuning methods for the machine learning algorithm such as grid search, gradient search [2], random search [3], and Bayesian optimization based methods [13, 25]. Naturally, evolutionary algorithms have also been applied to the hyperparameter optimization problems [] [**?**]. In the machine learning community, Bayesian optimization is often used and have shown good performance in several datasets. Bayesian optimization is a global optimization method of black-box and noisy objective functions, which maintains a surrogate model learned by using previously evaluated solutions. A Gaussian process is usually adopted as the surrogate model [25], which can easily handle the uncertainty and noise of the objective function. Bergstra et al. [5] have proposed the Tree-structured Parzen estimator (TPE) and shown better results than manual search and random search [3]. They have also proposed a meta-modeling approach [4] based on the TPE for supporting automatic hyperparameter optimization. Snoek et al. [26] used a deep neural network instead of the Gaussian process to reduce the computational cost for the surrogate model building and succeeded to improve the scalability.

The hyperparameter optimization approach often tunes the predefined hyperparameters such as the number of layers and neurons, and the type of activation functions. While this method has seen success, it is hard to design more flexible architectures from scratch.

2.2 Evolutionary Neural Networks

Evolutionary algorithms have been used to optimize the neural network architectures so far [23, 29]. The traditional approaches are not suitable for the model deep neural network architecture design since they usually optimize the number and connectivity of low-level neurons.

Recently, Fernando et al. [6] have proposed the differentiable pattern-producing networks (DPPNs) to optimize weights of a denoising autoencoder. The DPPN is a differentiable version of the compositional pattern-producing networks (CPPNs) [27]. This method focuses on the effectiveness of the indirect coding for weight optimization. That is, the general structure of network should be predefined.

Verbancsics et al. [31, 32] have designed the artificial neural networks and CNN architectures with the hypercube-based neuroevolution of augmenting topologies (HyperNEAT) [28]. However, to the best of our knowledge, these networks designed with HyperNEAT have failed to match the performance of state-of-the-art methods [Note (Shinichi): I do not know the contents of Verbancsics's papers]. Also, these methods deal with the architectures defined by human experts. Thus it is hard to design neural network architectures from scratch.

2.3 Reinforcement Learning Approach

The interesting approaches, automatic designing the deep neural network architecture using reinforcement learning, have been attempted recently [1, 35]. These studies showed that a reinforcement learning based method could construct the competitive CNN architectures for image classification tasks. In [35], a recurrent neural network (RNN) is used to generate the neural network architectures, and the RNN is trained with reinforcement learning to maximize the expected accuracy on a learning task. This method uses distributed training and asynchronous parameter updates with 800 GPUs to accelerate the reinforcement learning process. Baker et al. [1] have proposed a meta-modeling approach based on reinforcement learning to produce the CNN architectures. A Q-learning agent explores and exploits a space of model architectures with an ϵ−greedy strategy and experience replay.

These approaches adopt the indirect coding scheme for the network representation, which optimizes generative rules for the network architectures such as the RNN. Unlike these approaches, our

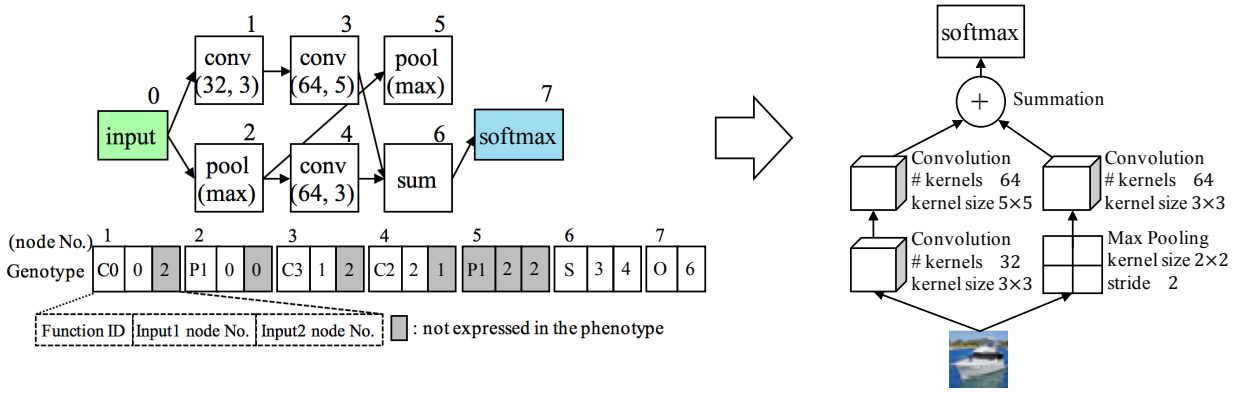

Figure 2: Example of a genotype and a phenotype. The genotype (Left) defines a CNN architecture (Right).

approach uses the direct coding based on Cartesian genetic programming to design the CNN architectures. Besides, we introduce the relatively highly-functional modules such as convolutional blocks and tensor concatenations to find better CNN architectures efficiently.

3 CNN ARCHITECTURE DESIGN USING CARTESIAN GENETIC PROGRAMMING

Our method directly encodes the CNN architectures based on CGP and uses the highly-functional modules as the node functions. The CNN architecture defined by the CGP is trained using training dataset, followed by the validation accuracy is assigned as the fitness of the architecture. Then the architecture is optimized to maximize the validation accuracy by the evolutionary algorithm.

In this section, we describe the network representation, the node functions, and the evolutionary algorithm used in the proposed method.

[Note (Shinichi): Checked up to here.]

3.1 Representation of CNN Architectures

We use a feedforward network that consists of R rows by C columns, i.e., the number of intermediate nodes of the network is $R \times C$. Figure 2 shows an example of the network of two rows by three columns and its genotype and phenotype.

The CGP inter-connectivity is determined by the levels back parameter l, which decides nodes of how many previous columns to connect to nodes in the current column, i.e., each node can have one or two connections to nodes in the l previous columns. Note that nodes in the same column are prohibited to be connected to each other. The types and connections of nodes are optimized by the evolutionary algorithm.

3.2 Node Functions

A type of each node is represented as four different types of layers: convolution, pooling, summation, and concatenation. The detailed parameters for each layer are shown in Table 1. The convolution node consists of a standard convolution processing with batch normalization [14] and rectified linear units [22]. The pooling node consists of the standard max pooling and average pooling

Table 1: Parameter details for each layer

Node type	Parameters	Values or description
Convolution	Kernel size	$\in \{3 \times 3, 5 \times 5\}$
	# kernels	$\in \{32, 64, 128\}$
	Stride	1
	# inputs	1
Pooling	Kernel size	2×2
	Stride	2
	# inputs	1
Summation	# inputs	2
		Element-wise addition.
Concatenation	# inputs	2
		Concatenate in the depth dimension.

processing. The summation node performs element-wise addition of two feature maps, channel by channel. If input feature maps to be added have different sizes, we pad the small feature map with zeros for increasing dimensions. The concatenation node concatenates two feature maps in the depth dimensions. If input feature maps to be concatenated are different sizes, we pad the small feature map with zeros. Adding these summation and concatenation nodes to the network architecture allow our method to generate shortcut connections or branching layers such as GoogleNet [30] and Residual Net [10].

3.3 Evolutionary Algorithm

We use mutation as a genetic operator. The mutation operator is applied to one individual as follows:

- Select several nodes with probability ϵ for each node.
- Change the type and connectivity of the selected nodes randomly.

Note that we apply the mutation operator to nodes until one or more active nodes that are associated with the output path are changed. As training a CNN can take hours, we apply the above mutation operator to save time.

Table 2: Parameter settings for the CGP

Parameters	Values or description
# Generations	300
Mutation rate	0.05
# Rows	5
# Columns	30
Levels back	10
Generation alternation model	(1 + 2) Evolution Strategy

We use a simple form of $(1 + \lambda)$ evolutionary strategy (with $\lambda = 2$) to design CNN architectures. The algorithm is as follows:

1. Generate an initial individual at random as a parent M, and train the CNN defined by M.
2. Generate a set of two offsprings C by applying the mutation operation to M.
3. Train these two CNNs defined by offsprings C.
4. Select an elite individual from the set of M and C, and replace M with the elite individual.
5. Return to step 2 until a stopping criterion is satisfied.

We employ the accuracy of the CNN on a validation set as the fitness of each individual.

4 EXPERIMENTS AND RESULTS

4.1 Dataset

We test our method to the image classification task with CIFAR-10 dataset which has 10 classes with 50,000 training images and 10,000 test images. We sampled 5,000 images randomly from the training set for the validation set, the remaining 45,000 images are used as the training set. In this experiment, we use data preprocessing and augmentation procedure. We first preprocess the data with the per-pixel mean subtraction. We then pad 4 pixels on each side, and choose a random 32×32 crop from the padded image. At last, we perform random horizontal flips on the cropped 32×32 image.

4.2 Training details

Parameter settings for the CGP is shown in Table 2. Once the CGP samples a CNN architecture, every CNN is trained for 50 epochs with the Adam optimizer [15] with an initial learning rate of 0.01. We reduce the learning rate by a factor of 10 every 20 epochs. All weights of CNN are initialized with He [9], and the mini-batch size is set to 128.

After finishing the evolution of the CGP, we select the architecture that showed the best validation accuracy. We then fine-tune the best model with a different training schedule. During this phase, we set a learning rate to 0.01 for 50 epochs, then 0.001 for 50 epochs. The other parameters are unchanged. After training the best model, we apply the model to the test set.

4.3 Results

[1]We implemented the VGG net [24] and applied the model to the CIFAR-10 dataset since the VGG net is not applied to the CIFAR-10 dataset in [24]. The architecture of the VGG is identical with the configuration D in [24]. We denote this model as VGG in this paper.

Table 3: Comparison of error rate on CIFAR-10 dataset.

Model	Error rate	# params
Maxout []	9.38	-
MetaQNN [1] *	9.09	11.11×10^6
Network in Network []	8.81	-
CGP2CNN (A)	8.17	
VGG [24] [1]	7.94	
CGP2CNN (B)	7.74	
ResNet [10]	6.61	1.7×10^6
Neural Architecture Search [35]	5.50	4.2×10^6

*The mean error rate of top 5 models.

Table 4: Comparison of error rate with the small CIFAR-10 dataset.

Model	Error rate
VGG [24]	27.67
ResNet [10]	24.10
CGP2CNN (A)	22.82
CGP2CNN (B)	–

We compare the classification performance of our method with the state-of-the-art-methods and summarize the results in Table 3. We refer to our architecture designed using functions in Table 1 as CGP2CNN (A) and the architecture designed using ResBlock as CGP2CNN (B). As can be seen in Table 3, our method can design architectures that are competitive with the state-of-the-art-methods. [Note (Suganuma): the rest of sentence is depending on experiment results...]

4.4 Designing architectures with a small dataset

To test the robustness of our method, we design a architecture with a small CIFAR-10 dataset that consists of 5,000 images randomly chosen from the CIFAR-10 dataset. We sampled 500 images randomly from the small training set for the validation set, the remaining 4,500 images are used as the training set. For settings of our method, the number of generation is set to 2,000. The other experimental settings are the same in the preceding section. We compare our method with the VGG and ResNet. For the VGG settings, the VGG is trained for 200 epochs with SGD with an initial learning rate of 0.01. We reduce the learning rate by a factor of 10 every 50 epochs. The mini-batch size is set to 128. For the ResNet, we train the model for 500 epochs. We use SGD with the mini-batch size of 128. We start with a learning rate of 0.1, then divide it by 10 at 250 and 375 epochs.

Table 4 shows that even with a small dataset, our model can find better architectures than the VGG and ResNet. The best architecture is shown Figure **??**. [Note (Suganuma): the rest of sentence is depending on experiment results...]

5 CONCLUSIONS

REFERENCES

[1] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. 2016. Designing Neural Network Architectures using Reinforcement Learning. *arXiv preprint arXiv:1611.02167* (2016).

[2] Yoshua Bengio. 2000. Gradient-based optimization of hyperparameters. *Neural computation* 12, 8 (2000), 1889–1900.

[3] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13, Feb (2012), 281–305.

[4] James Bergstra, Daniel Yamins, and David D. Cox. 2013. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. *ICML (1)* 28 (2013), 115–123.

[5] James S. Bergstra, Remi Bardenet, Yoshua Bengio, and Balazs Kegl. 2011. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*. 2546–2554.

[6] Chrisantha Fernando, Dylan Banarse, Malcolm Reynolds, Frederic Besse, David Pfau, Max Jaderberg, Marc Lanctot, and Daan Wierstra. 2016. Convolution by evolution: Differentiable pattern producing networks. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*. ACM, 109–116.

[7] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[8] Simon Harding. 2008. Evolution of image filters on graphics processor units using cartesian genetic programming. In *IEEE Congress on Evolutionary Computation*. IEEE, 1921–1928.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*. 1026–1034.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[11] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and others. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine* 29, 6 (2012), 82–97.

[12] Gao Huang, Zhuang Liu, Kilian Q. Weinberger, and Laurens van der Maaten. 2016. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993* (2016).

[13] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*. Springer, 507–523.

[14] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[15] Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *International Conference on Learning Representaions* (2015), 2452–2459.

[16] Alex Krizhevsky. 2009. Learning multiple layers of features from tiny images. *Technical report* (2009).

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

[18] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[19] Julian F. Miller and Peter Thomson. 2000. Cartesian genetic programming. In *European Conference on Genetic Programming*. Springer, 121–132.

[20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *Advances in neural information processing systems Workshop on Deep Learning* (2013).

[21] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, and others. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533. http://www.nature.com/nature/journal/v518/n7540/abs/nature14236.html

[22] Vinod Nair and Geoffrey E. Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*. 807–814.

[23] J. David Schaffer, Darrell Whitley, and Larry J. Eshelman. 1992. Combinations of genetic algorithms and neural networks: A survey of the state of the art. In *Combinations of Genetic Algorithms and Neural Networks, 1992., COGANN-92. International Workshop on*. IEEE, 1–37.

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

[25] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*. 2951–2959.

[26] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Md Mostofa Ali Patwary, Mr Prabhat, and Ryan P. Adams. 2015. Scalable Bayesian Optimization Using Deep Neural Networks.. In *ICML*. 2171–2180.

[27] Kenneth O. Stanley. 2007. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines* 8, 2 (2007), 131–162.

[28] Kenneth O. Stanley, David B. D'Ambrosio, and Jason Gauci. 2009. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life* 15, 2 (2009), 185–212.

[29] Kenneth O. Stanley and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation* 10, 2 (2002), 99–127.

[30] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1–9.

[31] Phillip Verbancsics and Josh Harguess. 2013. Generative neuroevolution for deep learning. *arXiv preprint arXiv:1312.5355* (2013).

[32] Phillip Verbancsics and Josh Harguess. 2015. Image Classification Using Generative Neuro Evolution for Deep Learning. In *Applications of Computer Vision (WACV), 2015 IEEE Winter Conference on*. IEEE, 488–493.

[33] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2015. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3156–3164.

[34] Richard Zhang, Phillip Isola, and Alexei A. Efros. 2016. Colorful image colorization. In *European Conference on Computer Vision*. Springer, 649–666.

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