A Genetic Programming Approach to Designing Convolutional Neural Network Architectures

Masanori Suganuma[1,2], **Shinichi Shirakawa**[3] and **Tomoharu Nagao**[3]
[1] RIKEN Center for AIP
[2] Tohoku University
[3] Yokohama National University
suganuma@vision.is.tohoku.ac.jp, shirakawa-shinichi-bg@ynu.ac.jp, nagao@ynu.ac.jp

Abstract

We propose a method for designing convolutional neural network (CNN) architectures based on Cartesian genetic programming (CGP). In the proposed method, the architectures of CNNs are represented by directed acyclic graphs, in which each node represents highly-functional modules such as convolutional blocks and tensor operations, and each edge represents the connectivity of layers. The architecture is optimized to maximize the classification accuracy for a validation dataset by an evolutionary algorithm. We show that the proposed method can find competitive CNN architectures compared with state-of-the-art methods on the image classification task using CIFAR-10 and CIFAR-100 datasets.

1 Introduction

Deep neural networks (DNNs) have shown excellent performance on many challenging machine learning tasks, such as image recognition, speech recognition, and reinforcement learning tasks. Convolutional neural networks (CNNs) [LeCun *et al.*, 1998], the DNN model often used for computer vision tasks, have seen huge success, particularly in image recognition tasks in the past few years. A standard CNN architecture consists of several convolutions, pooling, and fully connected layers. Recent studies have proposed a new CNN architecture that achieves higher performance, e.g., GoogLeNet [Szegedy *et al.*, 2015] and ResNet [He *et al.*, 2016]. These widely-used networks are designed by humans, but designing such architectures requires expert knowledge and trial-and-error. Hence, the automatic design of CNN architectures has much attention.

Recent studies for the automatic design of CNN require many weight-parameter optimizations; the weights of candidate CNN architectures are trained by a stochastic gradient descent (SGD), and the architectures are evaluated with trained weights on a validation dataset[1]. Evolutionary computation (EC) or reinforcement learning (RL) is employed as an architecture search method. The EC based architecture search methods [Miikkulainen *et al.*, 2017; Real *et al.*, 2017; Xie and Yuille, 2017] generate a new architecture by recombination and mutation operators, and evaluate it on a validation dataset after the weight training. The EC algorithms are expected to find the architecture that maximizes the performance (e.g., classification accuracy). The RL based architecture search methods [Baker *et al.*, 2017; Zoph and Le, 2017] optimize the policy that generates the CNN architectures by such as the policy gradient and Q-learning. Also, the reward used in the RL algorithms is the performance on a validation dataset. Reducing the computational cost for the architecture search is crucial topic because these methods require much computational cost to optimize neural network architectures.

In this work, we attempt to design CNN architectures based on genetic programming. We use the Cartesian genetic programming (CGP) [Miller and Thomson, 2000] encoding scheme to represent the CNN architecture, where the architecture is represented by a directed acyclic graph. The advantage of this representation is its flexibility; it can represent variable-length network architectures and skip connections. In addition, we adopt relatively highly-functional modules such as convolutional blocks and tensor concatenation as the node functions in the graph to reduce the search space of architectures. To evaluate the architecture represented by CGP, we train the network by SGD on a training dataset. The performance on a validation dataset is then assigned as the fitness of the architecture. Based on this fitness evaluation, the architecture is optimized to maximize the fitness (i.e., the validation accuracy) by evolutionary algorithms. To verify the performance of the proposed approach, we experimented involving constructing a CNN architecture for the image classification task with the CIFAR-10 and CIFAR-100 datasets. The experimental result shows that the proposed method can be used to find competitive CNN architectures compared with

This paper is an abridged version of the paper [Suganuma *et al.*, 2017a] that won a best-paper award at the evolutionary machine learning (EML) track of GECCO 2017 conference.

[1] In contrast, the traditional evolutionary neural networks, so-called neuroevolution [Stanley and Miikkulainen, 2002], typically optimizes both of the weights and architecture at the same time by an evolutionary computation method.

state-of-the-art models automatically.

2 CNN Architecture Design Using Cartesian Genetic Programming

2.1 Representation of CNN Architectures

We represent the architecture of CNNs by a directed acyclic graph defined on a two-dimensional grid of M rows and N columns. This graph is optimized by an evolutionary algorithm, where the graph called phenotype is encoded by a list of integers called genotype. Figure 1 shows an example of a genotype (top), a phenotype (middle) and the corresponding CNN architecture (bottom).

A genotype consists of $MN + 1$ genes, and each gene has information regarding the type T and connections C of a node. The node corresponds to a highly-functional module in CNN, e.g., the type T specifies one of the convolution or residual blocks, pooling layers, and tensor operations described later in detail. The connections C, specifying the node number in the anterior columns than the target node, represent which nodes are the input to the target node. This connection restriction ensures the feed-forward network structure. The last $(MN + 1)$-st gene, having only connection information C, represents the output layer, and its type is fixed based on the task.

In the CGP encoding scheme, there may be the nodes that do not connect to the output node; we call these nodes non-active nodes. In contrast, we define the nodes that connect to the output node as active nodes. Note that nodes depicted in the neighboring two columns are not necessarily connected. Thus, the resulting CNN architecture can have a different number of modules (layers) depending on the node connections, i.e., the number of layers is decided by evolutionary algorithms. Note that the maximum depth of a network is N. To control how the number of layers will be chosen, we introduce a hyper-parameter called level-back L, such that nodes given in the n-th column are allowed to be connected from nodes given in the columns ranging from $\max(0, n - L)$ to $n - 1$, where the zeroth column indicates the inputs. If we use smaller L, then the resulting CNNs will tend to be deeper.

2.2 Node Functions

Referring to the modern CNN architectures, we select the highly-functional modules as node functions. We prepare the six types of node functions called ConvBlock, ResBlock, max pooling, average pooling, concatenation, and summation.

The ConvBlock consists of standard convolution processing with a stride of 1 followed by batch normalization [Ioffe and Szegedy, 2015] and rectified linear units (ReLU) [Nair and Hinton, 2010]. The ResBlock is composed of a convolution processing, batch normalization, ReLU, and tensor summation. The ResBlock performs an identity mapping by shortcut connections as described in [He *et al.*, 2016]. For the number F and receptive field size k of filters of ConvBlock and ResBlock, we chose them from $F \in \{32, 64, 128\}$ and $k \in \{3 \times 3, 5 \times 5\}$, respectively.

The max and average poolings perform a max and average operation, respectively, over the local neighbors of the feature maps. We use the pooling with the 2×2 receptive field size and the stride of 2.

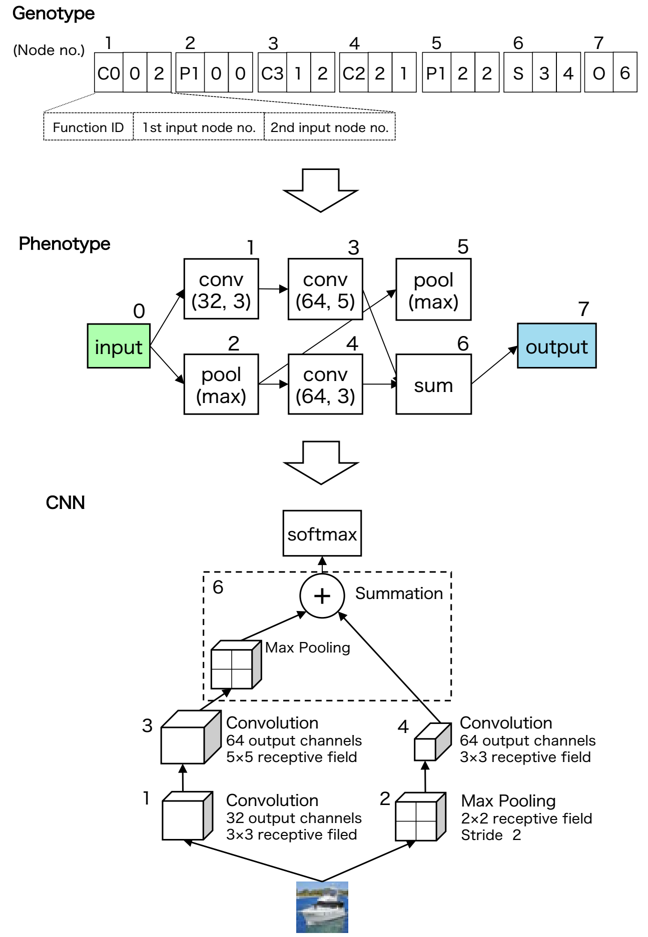

Figure 1: Example of a genotype, a phenotype, and the corresponding CNN architecture. The genotype (top) has information regarding the CNN architecture (bottom). In this example, the phenotype (middle) is defined on two rows and three columns.

The concatenation function concatenates two feature maps in the channel dimension. If the input feature maps to be concatenated have different numbers of rows or columns, we down-sample the larger feature maps by max pooling so that they become the same sizes of the inputs. The summation performs the element-wise addition of two feature maps, channel by channel. In the same way as the concatenation, if the input feature maps to be added have different numbers of rows or columns, we down-sample the larger feature maps by max pooling. In addition, if the inputs have different numbers of channels, we pad the smaller feature maps with zeros for increasing channels. In Figure 1, the summation node performs max pooling to the first input so as to get the same input tensor sizes.

The output node represents the softmax function with the number of classes. The outputs fully connect to all elements of the input.

2.3 Evolutionary Algorithm

We use a point mutation as the genetic operator in the same way as the standard CGP. The type and connections of each node randomly change to valid values according to a mutation rate. The standard CGP mutation has the possibility of affecting only non-active nodes. In that case, the phenotype does not change by the mutation and does not require a fitness evaluation again. The fitness evaluation of the CNN architectures is so expensive because it requires the training of the CNN. To use the computational resource efficiently, we apply the mutation operator until at least one active node changes for reproducing the candidate solution. We call this mutation a forced mutation. Moreover, to maintain a neutral drift, which is effective for CGP evolution [Miller and Smith, 2006; Miller and Thomson, 2000], we change only the non-active nodes of a parent by the mutation if the fitnesses of the offsprings do not improve. We call this mutation a neutral mutation.

We use the modified $(1 + \lambda)$ evolutionary strategy (with $\lambda = 2$ in our experiments). The procedure of our modified algorithm is as follows:

1. Generate an initial individual at random as parent P, and train the CNN represented by P followed by assigning the validation accuracy as the fitness.
2. Generate a set of λ offsprings O by applying the forced mutation to P.
3. Train the λ CNNs represented by offsprings O in parallel, and assign the validation accuracies as the fitness.
4. Apply the neutral mutation to parent P.
5. Select an elite individual from the set of P and O, and then replace P with the elite individual.
6. Return to step 2 until a stopping criterion is satisfied.

3 Experiments and Results

3.1 Dataset

We test our method on the image classification task using the CIFAR-10 and CIFAR-100 datasets in which the number of classes is 10 and 100, respectively. The numbers of training and test images are $50,000$ and $10,000$, respectively, and the size of images is 32×32. We randomly sample $45,000$ images from the training set to train the CNN, and we use the remaining $5,000$ images for the validation set of the CGP fitness evaluation.

We preprocess the data with the per-pixel mean subtraction. To prevent overfitting, we use a weight decay with the coefficient 1.0×10^{-4} and data augmentation. We use the data augmentation method based on [He *et al.*, 2016]: padding 4 pixels on each side followed by choosing a random 32×32 crop from the padded image, and random horizontal flips on the cropped 32×32 image.

3.2 Experimental Setting

To assign the fitness to the candidate CNN architectures, we train the CNN by SGD with a mini-batch size of 128. The softmax cross-entropy loss is used as the loss function. We initialize the weights by the He's method [He *et al.*, 2015] and use the Adam optimizer [Kingma and Ba, 2015] with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 1.0 \times 10^{-8}$, and an initial learning rate of 0.01. We train each CNN for 50 epochs and reduce the learning rate by a factor of 10 at 30th epoch.

For the parameter setting for CGP, we chose the mutation rate as $r = 0.05$, the graph with $M = 5$ and $N = 30$, and level-back $L = 10$. The offspring size of λ is set to two; that is the same number of GPUs in our experimental machines. We test two node function sets called ConvSet and ResSet for our method. The ConvSet contains ConvBlock, max pooling, average pooling, concatenation, and summation, and the ResSet contains ResBlock, max pooling, average pooling, concatenation, and summation. The difference between these two function sets is whether we adopt ConvBlock or ResBlock. The numbers of generations are 500 for ConvSet, 300 for ResSet.

After the CGP process, we re-train the best CNN architecture using $50,000$ training images, and we calculate the classification accuracy for the $10,000$ test images to evaluate the designed CNN architectures.

In this re-training phase, we optimize the weights of the obtained architecture for 500 epochs with a different training procedure; we use SGD with a momentum of 0.9, a mini-batch size of 128, and a weight decay of 5.0×10^{-4}. We start a learning rate of 0.01 and set it to 0.1 at 5th epoch, then we reduce it to 0.01 and 0.001 at 250th and 375th epochs, respectively. This learning rate schedule is based on the reference [He *et al.*, 2016]. We run the proposed method on the machine with 3.2GHz CPU, 32GB RAM, and two NVIDIA GeForce GTX 1080 (or two GTX 1080 Ti) GPUs.

3.3 Results

We run our method for three times on each dataset and report the classification performance. We compare the classification performance of our method with the state-of-the-art methods and summarize the classification error rates in Table 1. We refer to the architectures designed by our method as CGP-CNN. For instance, CGP-CNN (ConvSet) means the proposed method with the node function set of ConvSet. The models, Network in Network, VGG, ResNet, FractalNet, and Wide ResNet are hand-crafted CNN architectures, whereas the CoDeepNEAT, MetaQNN, Genetic CNN, Large-scale Evolution, and Neural Architecture Search are the models constructed by the architecture search method.

As can be seen in Table 1, the error rates of our methods are competitive with the state-of-the-art methods. In particular, CGP-CNN (ResSet) shows good performance, and the architectures constructed by using our method have a good balance between classification errors, the numbers of parameters and GPUs. The Neural Architecture Search achieved the best error rate on the CIFAR-10 dataset, but this method used 800 GPUs for the architecture search. Our method can find a competitive architecture with a reasonable machine resource.

Examples of CNN architectures designed by our method are shown in [Suganuma *et al.*, 2017a; 2017b]. We can observe that these architectures are quite different; the concatenation and summation nodes are not frequently used in CGP-CNN (ResSet), whereas these nodes are frequently used in

Model	# params	# GPUs	CIFAR-10	CIFAR-100
Network in Network [Lin *et al.*, 2014]	–	–	8.81	35.68
VGG [Simonyan and Zisserman, 2015]	15.2M	–	7.94	33.45
ResNet [He *et al.*, 2016]	1.7M	–	6.61	32.40
FractalNet [Larsson *et al.*, 2017]	38.6M	–	5.22	23.30
Wide ResNet [Zagoruyko and Komodakis, 2016]	36.5M	–	4.00	19.25
CoDeepNEAT [Miikkulainen *et al.*, 2017]	–	–	7.3	–
MetaQNN [Baker *et al.*, 2017]	3.7M	10	6.92	27.14
Genetic CNN [Xie and Yuille, 2017]	–	10	7.10	29.03
Genetic CNN from Wide Resnet [Xie and Yuille, 2017]	–	10	5.39	25.12
Large-Scale Evolution [Real *et al.*, 2017]	5.4M	250	5.40	–
Large-Scale Evolution [Real *et al.*, 2017]	40.4M	250	–	23.0
Neural Architecture Search [Zoph and Le, 2017]	37.4M	800	3.65	–
CGP-CNN (ConvSet)	1.5M	2	5.92 (6.70)	–
CGP-CNN (ConvSet)	2.0M	2	–	26.7 (27.9)
CGP-CNN (ResSet)	1.7M	2	5.01 (6.00)	–
CGP-CNN (ResSet)	4.6M	2	–	26.5 (27.6)

Table 1: Comparison of the error rates (%) and the number of learnable weight parameters on the CIFAR-10 and 100 datasets. We run the proposed method for three times for each dataset and report the classification errors in the format of "best (mean)." We refer to the architectures constructed by the proposed method as CGP-CNN. In CGP-CNN, the numbers of learnable weight parameters of the best architecture are reported. The values of other models except for VGG and ResNet for CIFAR-100 are referred from the literature.

CGP-CNN (ConvSet). These nodes lead a wide network; therefore, the network of CGP-CNN (ConvSet) is a wider structure than that of CGP-CNN (ResSet).

4 Related Work

Some architecture search methods based on either EC [Miikkulainen *et al.*, 2017; Real *et al.*, 2017; Xie and Yuille, 2017] or RL [Baker *et al.*, 2017; Zoph and Le, 2017] for DNNs have been proposed at nearly the same time with our work. Since these approaches require much computational cost in general, the recent works concentrate on the reduction of computational cost and improvement of architecture search efficiency [Cai *et al.*, 2018; Liu *et al.*, 2017; 2018; Pham *et al.*, 2018; Real *et al.*, 2018; Zhong *et al.*, 2017; Zoph *et al.*, 2017]. For instance, Cai et al. [Cai *et al.*, 2018] specify an initial architecture by an existing one, and the actions, deepening the architecture by adding a layer or widening an existing layer by replacing an existing layer with a wider layer, are applied to the initial architecture to generate a new architecture. Pham et al. [Pham *et al.*, 2018] have introduced the parameter sharing across models during the architecture search and achieved a test error of 2.89% on the CIFAR-10 dataset with one GPU.

Most methods of the CNN architecture search are applied to image classification tasks such as using CIFAR-10. They, however, can be naturally applied to other computer vision tasks and DNNs. The architecture search method explained in this paper has applied to automatically construct the autoencoder typed CNNs for the image denoising and inpainting tasks [Suganuma *et al.*, 2018]. The result shows that the designed architectures outperform the existing state-of-the-art hand-crafted architectures on both denoising and inpainting tasks.

5 Conclusion

In this work, we have attempted to take a GP-based approach for designing the CNN architectures and have verified its potential. The proposed method designs the CNN architectures based on CGP and adopts the highly-functional modules, such as convolutional blocks and tensor operations, for searching for the adequate architectures efficiently. We have constructed the CNN architecture for the image classification task with the CIFAR-10 and CIFAR-100 datasets. The experimental result showed that the proposed method can automatically find competitive CNN architectures compared with the state-of-the-art models.

One direction of future work is to develop the evolutionary algorithm to reduce the computational cost of the architecture design, e.g., increasing the training data for the neural network as the generation progresses. Another future work is to apply the proposed method to other image datasets and tasks.

References

[Baker *et al.*, 2017] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *ICLR*, 2017.

[Cai *et al.*, 2018] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *AAAI*, pages 2787–2794, 2018.

[He *et al.*, 2015] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, pages 1026–1034, 2015.

[He *et al.*, 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[Ioffe and Szegedy, 2015] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456, 2015.

[Kingma and Ba, 2015] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[Larsson *et al.*, 2017] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. In *ICLR*, 2017.

[LeCun *et al.*, 1998] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[Lin *et al.*, 2014] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. In *ICLR*, 2014.

[Liu *et al.*, 2017] Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. *arXiv:1712.00559*, 2017.

[Liu *et al.*, 2018] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. In *ICLR*, 2018.

[Miikkulainen *et al.*, 2017] Risto Miikkulainen, Jason Zhi Liang, Elliot Meyerson, Aditya Rawal, Dan Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, and Babak Hodjat. Evolving deep neural networks. In *GECCO*, 2017.

[Miller and Smith, 2006] Julian F. Miller and Stephen L. Smith. Redundancy and computational efficiency in cartesian genetic programming. *IEEE Transactions on Evolutionary Computation*, 10(2):167–174, April 2006.

[Miller and Thomson, 2000] Julian F. Miller and Peter Thomson. Cartesian genetic programming. In *EuroGP*, pages 121–132, 2000.

[Nair and Hinton, 2010] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814, 2010.

[Pham *et al.*, 2018] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[Real *et al.*, 2017] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka L. Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *ICML*, pages 2902–2911, 2017.

[Real *et al.*, 2018] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2018.

[Simonyan and Zisserman, 2015] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

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

[Suganuma *et al.*, 2017a] Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. A genetic programming approach to designing convolutional neural network architectures. In *GECCO*, pages 497–504, 2017.

[Suganuma *et al.*, 2017b] Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. A genetic programming approach to designing convolutional neural network architectures. *arXiv preprint arXiv:1704.00764*, 2017.

[Suganuma *et al.*, 2018] Masanori Suganuma, Mete Ozay, and Takayuki Okatani. Exploiting the potential of standard convolutional autoencoders for image restoration by evolutionary search. *arXiv preprint arXiv:1803.00370*, 2018.

[Szegedy *et al.*, 2015] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, pages 1–9, 2015.

[Xie and Yuille, 2017] Lingxi Xie and Alan L. Yuille. Genetic CNN. In *ICCV*, pages 1379–1388, 2017.

[Zagoruyko and Komodakis, 2016] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, pages 87.1–87.12, 2016.

[Zhong *et al.*, 2017] Zhao Zhong, Junjie Yan, and Cheng-Lin Liu. Practical network blocks design with Q-Learning. In *arXiv: 1708.05552*, 2017.

[Zoph and Le, 2017] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

[Zoph *et al.*, 2017] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *arXiv:1707.07012*, 2017.