Exploring Randomly Wired Neural Networks for Image Recognition

Ukho Shin
KAIST

Hyemin Lee
KAIST

Abstract

The manual design from simple chain-like models and structure with multiple wiring paths enabled development of neural networks for image classification. ResNets and DenseNets, one of the most famous CNN networks, are highly efficient because of their innovative wiring plans. To implement efficient networks, neural architecture search (NAS) automates the process of finding network architecture and operation types. However, the search space of possible wirings is still constrained and driven by manual design. In this project, we explore a more diverse set of connectivity patterns based on randomly wired neural networks. First, we define stochastic network generator that represents entire network generation process. Then, we used three mathematical random graph models and generate randomly wired graphs by converting graphs into networks. Several random generators produced random network instances that have competitive accuracy on the CIFAR-10 benchmark. This result suggests that new efforts focusing on designing better network generators may lead to new breakthroughs by exploring less constrained search spaces with more room for novel design

Keywords—NAS, randomly wired neural networks, network generator

I. RELATED WORK

A. Network Wiring

Early recurrent neural networks (RNNs) and convolutional neural networks (CNNs) sed chain-like wiring patterns.

What we call deep learning today descends from the connectionist approach to cognitive science [1]. The performance of a neural network highly depends on how computational networks are wired. Thus, neural network has developed from chain-like wiring structure to more complex connectivity patterns like ResNet and DenseNet so far.

To find efficient network structures, neural architecture search(NAS) (quotation) has emerged as a promising direction for jointly searching wiring patterns and which operations to perform. NAS method focus on search of operations and architecture of neural network while overlooking the important component which we are calling network generator. The NAS search space defines a set of operations (e.g. convolution, fully-connected, pooling) and how operations can be connected to form valid network architectures. However the design of NAS search space is usually handwritten like ResNet[4] and DenseNet[5]. Therefore, NAS search space of allowed siring patterns are still constrained. In this project, we tried to loosen this constraint and design novel network generation.

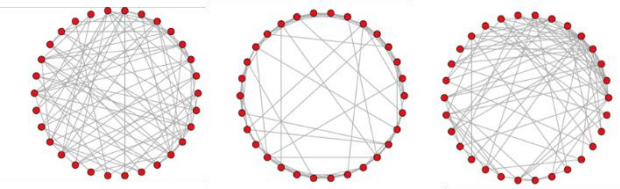

Figure 1. **Random graph.** From the left, each graphs are ER random graph network with (N = 30, <k> = 5.8), WS small world network with (N = 30, <k> = 6.0), and BA scale-free network with (N = 30, <k> = 5.7)

To reduce the bias from the hand designing process of network wiring, we used three sets of mathematical random graph models in graph theory : Erdos-R ″ enyi (ER)[6], Barabasi-Albert (BA)[8], Watts-Strogatz (WS)[7] models.

To make network model from random graphs, we made random graphs with directed acyclic graph (DAG) form, and apply a simple mapping from nodes to the neural network layers. We applied same type of convolution to all nodes since this project focus on network wiring design, not an operation optimization.

We note that the randomly wired networks are not free from parameters even they are random. Many strong priors affect network generator, including the choice of random graph model and the probability of connection between two nodes. Each random graph model (ER, BA, WS) has certain probabilistic behaviors such that generated graphs have similar properties. Therefore, the generator design by some priors determines a probabilistic property of neural networks.

This project, based on the paper Exploring Randomly Wired Neural Networks for Image Recognition [2], aims to cover the rest of NAS search space. Concurrent research on random search for NAS [3] show that random search is efficient in the NAS search space, which means NAS network generator in our work. The result of the studies implies prior induced by the NAS generator design tend to produce high performance models. This project goes further from the design of established NAS generators and explore mode diverse random generator designs.

Finally, this project suggests a new transition from designing an in individual network to designing a network generator. Rather than focusing primarily on search with a fixed generator, we suggest designing new network generators that produce new families of models for searching. Current NAS study is not been automated at all since designing network generator is still human-designed. Therefore, this project could establish network generator engineering instead of network engineering

II. Related work

A. Network Wiring

Early recurrent neural networks (RNNs) and convolutional neural networks (CNNs) sed chain-like wiring patterns.

Inception CNNs concatenate irregular branching pathways, and ResNets use residual block which includes skip connection which is represented in equation [1] and figure 2 as a regular wiring template. DenseNet use concatenation operation which is represented in equation [2] and figure 3.

$$x + F(x) \quad \text{(skip connection) [1]}$$
$$[x, F(x)] \quad \text{(concatenation) [2]}$$

The Inception, ResNet, and DenseNet wiring neural nets are effective in general cases.

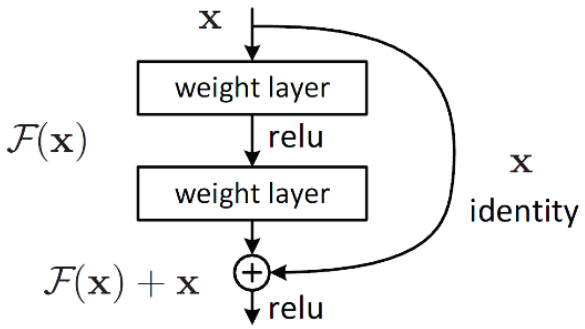

Figure 2**. Residual block.** Left figure is optimal Normal cell structure and right figure is optimal Reduction cell which are found by neural architecture search process/

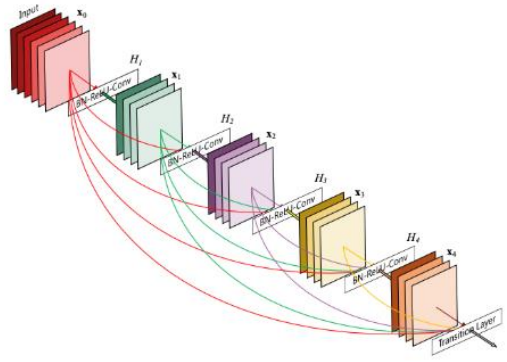

Figure 3. **DenseNet architecture.** Left figure is optimal Normal cell structure and right figure is optimal Reduction cell which are found by neural architecture search process/

B. Neural architecture search (NAS)

Neural architecture search is of exploring neural network structure using reinforcement learning. Zoph and Le [10] define a NAS search space and investigate reinforcement learning (RL) as an optimization algorithm.

Figure 4 shows the process of generating model description with a controller RNN. Controller RNN produces a parameter of CNN model like number of filters, filter height, filter width. After a CNN model made, validation accuracy is used as reward and optimized with policy gradient method.[9]

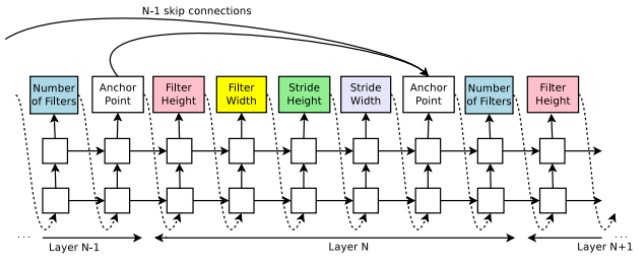

Figure 4. RNN controller predicts convolution layer. a batch of RNN outputs compose a layer of CNN

However, the original NAS process cannot be used widely since it is very slow to train and difficult to apply to other data. Therefore, the developed model, NASNet maintained the existing NAS method, but find a new search space which are Normal Cell and Reduction Cell. NASNet predicted the best structure of two cells and repeat cell to construct the network. Figure 5 represents the Optimal Cell from NASNet search space.

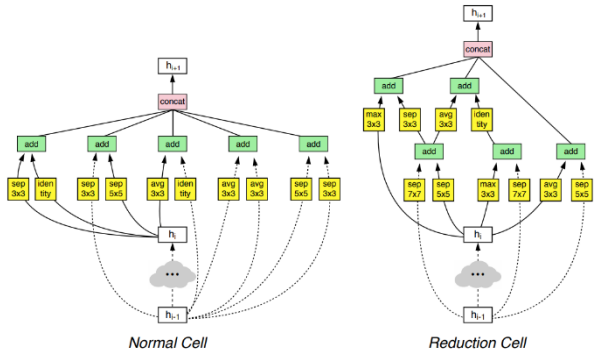

Figure 5. **Optimal Cell from NAS.** Left figure is optimal Normal cell structure and right figure is optimal Reduction cell which are found by neural architecture search process.

III. Methology

A. CIFAR-10 dataset

CIFAR-10 dataset was used instead of ImageNet dataset which was used in baseline paper. CIFAR-10 dataset contains 32 x 32 colour images of 50000 train data and test data. whole data have to divided into 10 groups.[11] This dataset is used to evaluate RandWire models and handwritten models..

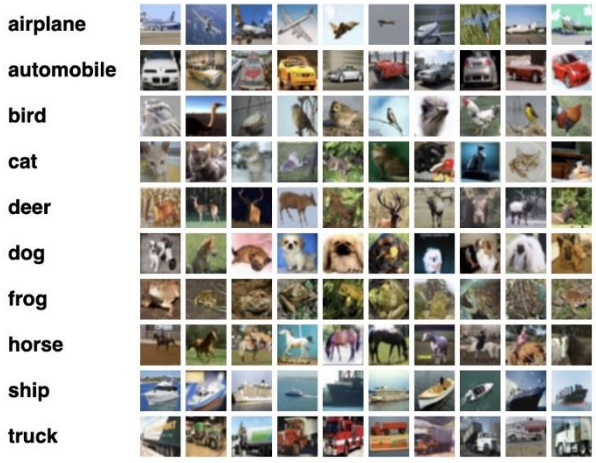

Figure 6. **CIFAR-10**. CIFAR-10 dataset is consist of 10 classes, which are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

B. Network Generators

In this project, we define a network generator as a mapping g from a parameter space theta to a space of neural network architectures. (citation). For a given Θ, g(Θ) returns a neural network instance n ∈ N. The set N is typically a family of related networks, for examples, VGG nets, ResNets, or DenseNets.

$$g : \Theta \longmapsto N \text{ (deterministic mapping) [1]}$$

Stochastic network generators. The above network generator g(Θ) produces a unique network architecture n when the value of theta is determined with same theta. By adding a random seed argument s to the original network generator g(Θ) , we can extend deterministic network generator. For a fixed value theta, changing the value of s = 1, 2, 3, … produce a random family of networks by calling g(Θ, s). We call this new network generator g(Θ, s) as stochastic network generator. In this project, we used random seed s as the probability of connection between two nodes to construct a random graph.

$$g : \Theta, s \longmapsto N \text{ (stochastic mapping) [2]}$$

C. Randomy Wired Neural Networks

The present research on NAS depends on hand design and human-defined priors. Therefore, current NAS studies failed to accomplish "AutoML" and still requires human effort.

For that reason, we will define network generators that produces neural networks for image recognition with mathematical random graphs which depend on human - determined parameters. To minimize the bias caused by human determination, we use 3 random classical random graph models. Our network generator is constructed by following steps.

Generating general graphs. Since our network generator is based on random graph, we first generate general graph. General graph consists of set of nodes and edges that connect nodes. We can freely use any general graph generator from ER/BA/WS since there is no restriction of how the graphs correspond to neural networks. Generated graph is mapped to a computable neural network.

The mapping from a general graph to neural network operations is human-designed, and we implemented a simple mapping process which is represented next. By that way, we can focus on graph wiring patterns.

Edge operations. Assuming the construction of graph is directed, we can map an edge into a data flow from one node to another node.

Node operations. A node in a directed graph can have some input edges and output edges connected to other nodes. We define some operations that applied to one node as following terms.

- Aggregation : The input data to a node combined by a weighted sum. The parameters of weighted sum are learnable and positive. Positive weights can be made by applying sigmoid on unrestricted weights.
- Transformation : The ReLU-convolution-BN triplet is applied to aggregated data. We use the form of 3 by 3 depth-wise convolution followed by 1 x 1 convolution, with non-linearity between. Since we do not focus on the operation, the same type of convolution is used for all nodes.
- Distribution : The same copy of the transformed data is sent out by the output edges of the node

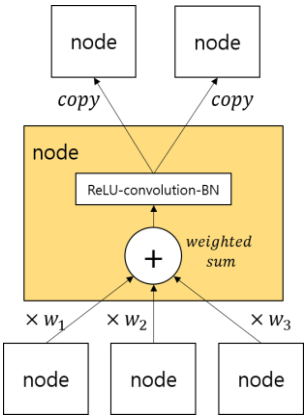

Figure 7. **Node operations** designed for our random graphs. We illustrate a node with yellow box with 3 input edges and 2 output edges. The aggregation step is done by weighted sum layer with positive weights w_1, w_2, w_3. The transformation step is a combination of ReLU, convolution-BN triplet, and we will simply denote as conv. The transformed data will be copied and send to 2 output nodes.

Input and output nodes. Thus far, since we have to make single input node and single output node to make a valid neural network, multiple input and output nodes yet have to be processed more.

To gather all the input nodes, we create additional single node that is connected to all original input nodes. This is the unique input node for given graph that sends out the same input data to all original input nodes.

Similarly, we create additional single extra node that is connected to all original output nodes. Additional node becomes unique output node and it combines all output node by computing average value of original output nodes.

These two nodes do not includes convolution operation since it is just for combining nodes. Later, we exclude these two nodes to count total node N.

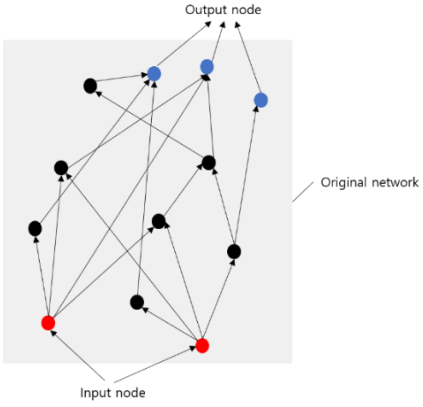

Figure 8. **Network with input and output node.** Original network is in the gray box. There exists two original input nodes which are colored with red, and three original ourput nodes which are colored with blue. We have to create total two node which combines original input nodes and orininal ourput nodes to unique input / output node.

Stages. We will divide our neural net into 5 convolution stages and classifier stage to progressively down-sample feature maps. One stage is generated from one graph. Each stages are connected to other stages by its unique input and output nodes. The channel count in a random graph is multiplied by 2 when stage is increased by one due to down-sampling.

Stage	Output	Small regime	Regular regime
conv1	48 × 48	3×3 conv, C/2	
conv2	24 × 24	3×3 conv, C	Random wiring N/2, C
conv3	12× 12	Random wiring N, C	Random wiring N, 2C
conv4	6 × 6	Random wiring N, 2C	Random wiring N, 4C
conv5	3× 3	Random wiring N, 4C	Random wiring N, 8C
classifier	1× 1	1× 1 conv, 1280-d Global average pool, 100-d fc, softmax	

Table 1. **RandWire architecture** for small and regular computation networks. A random graph is denoted by the node count (N) and channel count for each node (C). We use conv to denote a ReLU-Conv-BathNormalization triplet. The input size is 32 x 32 pixels. The change of the output size implies a stride of 2 (omitted in table) in convolutions that are right after the input of each stage.

Table 1 shows 2 kinds of architecture of randomly wired neural net, reffered to as RandWire, used in our project. First architecture is named small regime, which will be compared by simple models like MobileNet. Second architecture is named regular reigime, which will be compared by complex models like ResNet and DenseNet.

For some stages, we use single convolution layer for simplicity. The classifier stage is follow at the last of our RandWire model.

D. Random Graph Models

We use the three classical random graph models chosen in original paper to generate graphs, and use a simple heuristic to turn them into DAGs.

Erdös-Rényi (ER). In the ER model [6], with N nodes, an edge between two nodes is connected with probability P, independent of all other nodes and edges. This process is iterated for all pairs of nodes. The ER generation model has only a single parameter P, and is denoted as ER(P). Any graph with N nodes has non-zero probability of being generated by the ER model.

Barabási-Albert (BA). The BA model [8], generates a random graph by sequentially adding new nodes. The initial state is M nodes without any edges ($1 \leq M < N$). The method sequentially adds a new node with M new edges. For a node to be added, it will be connected to an existing node v with probability proportional to v's degree. The new node repeatedly adds non-duplicate edges in this way until it has M edges. Then this is iterated until the graph has N nodes. The BA generation model has only a single parameter M, and is denoted as BA(M). Any graph generated by BA(M) has exactly $M\cdot(N-M)$ edges. So the

Watts-Strogatz (WS). The WS model [7], was defined to generate small-world graphs [12]. Initially, the N nodes are regularly placed in a ring and each node is connected to its $K/2$ neighbors on both sides (K is an even number). Then, in a clockwise loop, for every node v, the edge that connects v to its clockwise i-th next node is rewired with probability P. "Rewiring" is defined as uniformly choosing a random node that is not v and that is not a duplicate edge. This loop is repeated $K/2$ times for $1 \leq i \leq K/2$. K and P are the only two parameters of the WS model, denoted as WS(K, P). Any graph generated by WS(K, P) has exactly N·K edges.

Converting undirected graphs into DAGs. The original graphs generated by ER, BA and WS models are undirected graphs. We implemented those algorithms for generating DAGs instead of undirected ones, using simple heuristic. We assign indices to all nodes in a graph, and set the direction of every edge as pointing from the smaller-index node to the larger-index one. The heuristic ensures that there is no cycle in the resulted directed graph. The node indexing strategies for the models are the following. ER: indices are assigned in a random order. BA: the initial M nodes are assigned indices 1 to M, and all other nodes are indexed following their order of adding to the graph. WS: indices are assigned sequentially in the clockwise order.

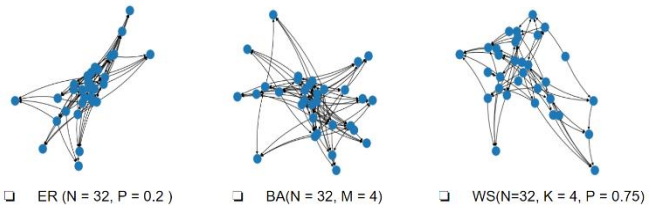

Figure 9. **Example randomly generated DAGs**. Representing nodes as blue circles and directed edges as arrows. Showed using matplotlib and networkx.

E. Design and optimization

Our randomly wired neural networks are generated by a stochastic network generator $g(\Theta, s)$. The optimization of graph generating parameters, P of ER(P), M of BA(M), (K, P) in WS(K, P) are only can be optimized by trial-and-error. So, we just compared the extracted accuracy of networks generated by each graph parametrized differently. WS models for P = 0.2, 0.4, 0.6, 0.8, BA models for M = 2, 3, 4, 5, WS models for combination of K = 2, 4, 6, 8 and P = 0.25, 0.5, 0.75.

IV. Experiments

A. Models Property

We conduct experiments on the ImageNet CIFAR-10 task, classifying images into 10 classes. Using 50K training images of 5K for each class, and 10K images of validation and test.

Architecture details. Our experiments span a small computation regime (e.g., MobilNet) and a regular computation regime (e.g. ResNet-50/101). RandWire nets in these regimes are in Table 1, where the number of nodes N and the number of channels C determine network complexity. We set N = 32, same as paper [1], and set C for the models to have similar numbers of parameters as well as other existing models, C = 60 for small regime, and C = 78 for the regular regime. The number of parameters of RandWire networks are about 5M for small regime, about 18M for large regime.

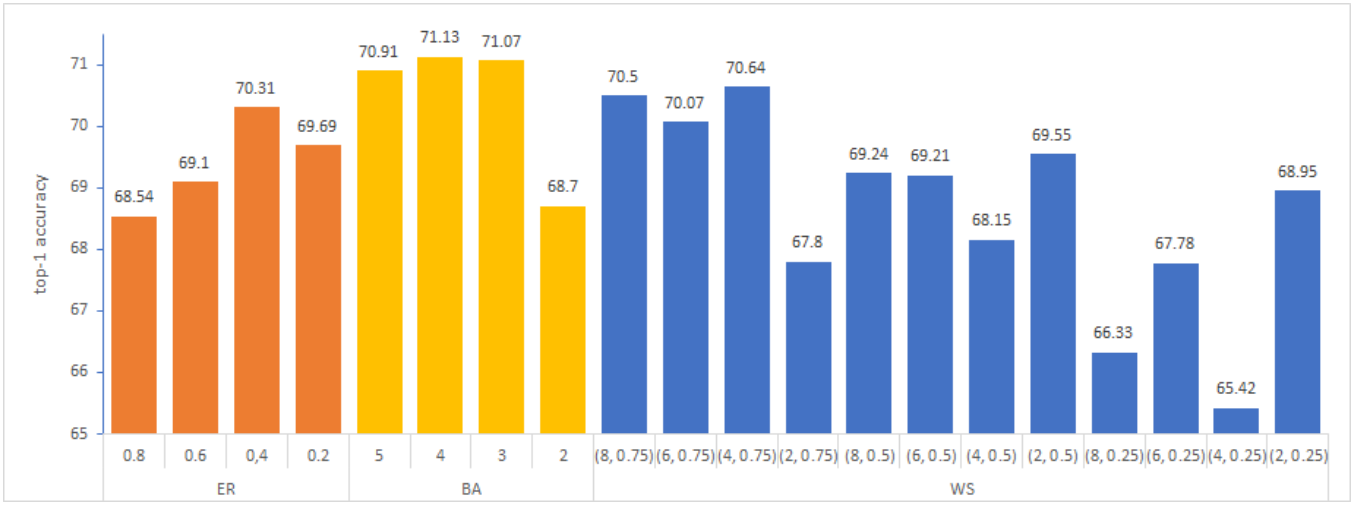

Figure 10. **Comparison on random graph generators: ER, BA, and WS** in the small computation regime. Each bar represents the result of a generator under a parameter setting for *P*, *M*, or (*K*, *P*) tagged in *x*-axis. The results are the top-1 accuracy within 50 epochs.

small regime models

Network	top-1 acc.	params(M)
MobileNet	0.7394	4.8
MobileNetV2	0.7456	4.1
MobileNetV3Small	0.6824	3.1
MobileNetV3Large	0.7414	6
NasNetMobile	0.6926	5.8
RandWire(BA(4), C = 60)	0.7113	4.7

regular regime models

Network	top-1 acc.	params(M)
ResNet50	0.7170	26.2
ResNet101	0.7262	45.2
VGG16	0.8027	15.8
VGG19	0.7885	21.1
RandWire(WS(4, 0.75) C = 78)	0.7910	17.8
RandWire(BA(4), C = 78)	0.7628	18.5

Table 2, Table 3. **Comparison on random graph networks and existing models** in small and regular computation regime. Showing top-1 accuracy within 50 epochs and the number of parameters.

Execute Models. By limitation of time and GPU performance, we could only train and evaluate each model once.

Implementation details. We train our RandWire networks and existing ones for 50 epochs. We use standard SGD optimizer using sparse categorical crossentropy for loss to train each model, learning rate or other parameters are set up as python tensorflow default values. We chose the best 1 accuracy during 50 epochs to evaluate and compare.

Existing Models. We used 'MobileNet', 'MobileNetV2', 'MobileNetV3Small/Large', and 'NasNetMobile' for small regime and did not used 'ShuffleNet' and 'SuffleNetV2' since they were not applicated in Keras. Also, for regular regime, we used 'ResNet50/101' and 'VGG16/19' to compare with our RandWire Models.

Random graph generators. Figure 10 compares the results of network generators of different graph models for each random stage in the small computation regime, having about 5M parameters.

We chose two random graph generators to execute in regular computation regime. BA(4) which has the best accuracy in our result of small regime, WS(4,0.75) which has the best accuracy in the paper [1].

Weighted Sum. There's no existing embodied layer for weighted sum of nodes, so we need to embody it in our hands. Just multiplying trainable values to nodes are not worked as the weight became non-trainable if it is not inside a form of layer class. So we implemented a custom layer to use it, using 'GlorotNormal' initializer for weight, and activation function 'ReLu' to make weights to be positive.

```
class Weighted_Sum(layers.Layer):
  def __init__(self, n_inputs):
    super(Weighted_Sum, self).__init__()
    self.ws = [self.add_weight(shape=(1,), initializer=keras.initializers.GlorotNormal()
                               ,trainable=True) for j in range(n_inputs)]
  def call(self, inputs):
    return sum([inputs[j] * tf.nn.relu(self.ws[j]) for j in range(len(inputs))])
```

Figure 11. **Customized weighted sum layer** with 'inputs' is a list of input nodes and 'n_inputs' is the number of 'inputs'. The layer is implemented by multiplying trainable variable to each node and sum.

B. Comparisons

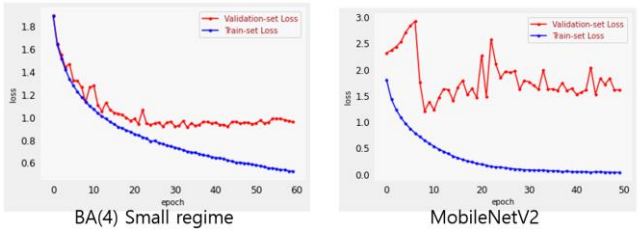

Figure 12. **Loss of train set and validation set** for RandWire BA(4) networks and MobileNetV2, calculated during 50 epochs of training.

Small computation regime. Table 2 compares our results in the small computation regime. RandWire with BA(4) has accuracy 71.1%. This result accuracy is normal compared to other ones, and not so good compared to existing models.

It is different from RandWire model in the paper [1], which WS(4,0.75) was selected as having the most highest accuracy than other Randwire models and overwhelmed other small regime existing models. But analyzing the graph in Figure 12, it's because of our short epochs. Since the highest performed model 'MobileNetV2' is performing well from the beginning and has gentle slope in training loss at about 50-th epoch and BA(4) reducing training loss slowly at first but still has steep slope at about 50th epoch. It means RandWire models can potentially be better than now, perhaps WS(4,0.75) can overcome BA(4) after more epochs. So we chose two RandWire models BA(4) and WS(4,0.75) to train and evaluate in the regular regime.

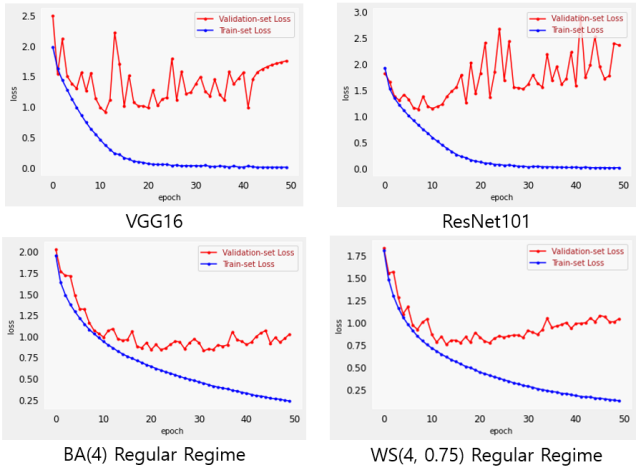

Figure 13. **Loss of train set and validation set** for RandWire BA(4), WS(4,0.75), ResNet101, and VGG16 models calculated during 50 epochs of training.

Regular computation regime. Table 3 compares our results in the regular computation regime. In this regime, as compared in the paper [1], WS(4,0.75) outperforms BA(4) and all the other existing models except VGG16.

For the similar reason with in small regime, the higher accuracy of VGG16 is guessed because of short epochs. The VGG performs very well from the beginning, even than ResNet too, and makes very gentle slope after about 20th epoch. But our RandWire models works still has steep slopes after 50th epoch, then about 1% of distance between VGG16 and WS model will be decrease more or reverse after epochs

V. Conclusion

Our project replicated the paper [1], exploring randomly wired neural networks driven by three random graph models. It is said to be replicated well, as our RandWire networks performed competitive with existing experts' hand-designed models. We learned and tested the concept of a network generator.

We replicated the functional domains, but couldn't replicate many parts of comparison and evaluation by the limitation of GPU performance we used, Google Colab, and time consumed. We could replicate the evaluation with testing each models several times, and unembodied experiments about graph damage, changing node operations, or other furthermore experiments like changing the number of stages or the strategies to generate random graphs, later if we get other GPUs with good performance.

VI. References

[1] Xie, Saining, et al. "Exploring randomly wired neural networks for image recognition." Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019.

[2] J. A. Fodor and Z. W. Pylyshyn. Connectionism and cognitive architecture: A critical analysis. Cognition, 28(1-2):3– 71, 1988.

[3] L. Li and A. Talwalkar. Random search and reproducibility for neural architecture search. arXiv:1902.07638, 2019..

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In CVPR, 2016.

[5] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In CVPR, 2017.

[6] P. Erdos and A. R ˝ enyi. On the evolution of random graphs. ´ Publ. Math. Inst. Hung. Acad. Sci, 5(1):17–60, 1960.

[7] D. J. Watts and S. H. Strogatz. Collective dynamics of smallworldnetworks. Nature, 393(6684):440, 1998.

[8] R. Albert and A.-L. Barabasi. Statistical mechanics of com- ´ plex networks. Reviews of modern physics, 74(1):47, 2002.

[9] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. In ICML, 2017.

[10] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. In ICML, 2017.

[11] Krizhevsky, Alex, and Geoffrey Hinton. "Learning multiple layers of features from tiny images." (2009): 7.

[12] M. Kochen. The Small world. Ablex Pub., 1989. 2, 5 [19] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet clas