Connectivity Properties of Neural Networks Under Performance-Resources Trade-off

Romuald A. Janik [*1] **Aleksandra I. Nowak** [*2]

Abstract

We analyze the structure of network architectures obtained when trained under a performance-resources trade-off for various datasets. To this end, we use a flexible setup allowing for a neural network to learn both its size and topology during the course of a standard gradient-based training. The resulting network has the structure of a graph tailored to the particular learning task and dataset. We explore the properties of the resulting network architectures for a number of datasets of varying difficulty observing systematic regularities. The obtained graphs can be therefore understood as encoding nontrivial characteristics of the particular classification tasks.

1. Introduction

Classical Deep Learning Networks typically have a fixed and rigid structure *a-priori* independent of the specific learning task. In addition, a network usually utilizes all of its resources irrespective of whether the given learning task is challenging or not. One might expect, however, that the *natural* network structure and size for a specific dataset should depend on the characteristics of the corresponding task.

The goal of this paper is to analyze the dependence between the dataset difficulty and the corresponding network architecture under a performance-resources trade-off. To this end, we adopt the construction introduced in (Xie et al., 2019) and expanded in (Janik & Nowak, 2020), in which the architectures are described given a Directed Acyclic Graph (DAG). In that formulation the graph's nodes represent the layers (or stacks of operations), while the edges define the connectivity patterns between the layers. We start from a fully-connected setup and allow the model to directly adjust its effective size and topology during the training. This is achieved by incorporating a very natural performance-resources trade-off in the network's loss function:

$$L_{total} = L_{objective} + L_{resources}, \tag{1}$$

where the $L_{resources}$ is an L_1 loss on the weights of the connections between the layers.

The aim of this paper is to *analyze* the graph architectures obtained in the above described framework when training the networks on a number of datasets of varying difficulty on an image recognition task. In particular we are interested in the following questions:

1. What are the common graph features of the resulting networks?
2. How do the properties of the network depend on the specific dataset and its difficulty?

Indeed, as we will see, the differences in the structure of the obtained networks go beyond just an overall change in size. Therefore, the studied framework has the benefit of quantifying, in a nontrivial way, the internal structure of a dataset or, more generally, the learning task.

2. Related work

The issue of introducing sparsity into the neural network under a performance-resources trade-off framework has been extensively explored in the field of neural network unstructured (LeCun et al., 1990; Han et al., 2015; Lin et al., 2017) and structured (Wen et al., 2016; Molchanov et al., 2016; Anwar et al., 2017; Li et al., 2016) pruning – see (Cheng et al., 2017; Blalock et al., 2020) for a survey. Recently, (Frankle & Carbin, 2018) hypothesize that pruning is merely a way of retrieving an optimal subnetwork for a given input weight initialization, while (Liu et al., 2018b) argue that perhaps the structurally-pruned architecture itself is most important for the efficiency of the final model. The link between the neural network architecture and dataset is also at the key interest in Neural Architecture Search (Zoph & Le, 2017; Baker et al., 2016; Real et al., 2019; Liu et al., 2018a; Ying et al., 2019; Zhang et al., 2018; Elsken et al., 2019). In particualr, (Xie et al., 2019; Janik & Nowak, 2020)

[*]Equal contribution [1]Institute of Theoretical Physics, Jagiellonian University, Łojasiewicza 11, 30-348 Kraków, Poland [2]Faculty of Mathematics and Computer Science, Jagiellonian University, Łojasiewicza 6, 30-348 Kraków, Poland.. Correspondence to: Aleksandra Nowak <nowak.aleksandrairena@gmail.com>.

DyNN workshop at the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, 2022.

explore wiring topologies based on random graphs. In this work we are not interested in designing any new pruning or NAS approach. Instead, we analyze the properties of the network connectivity patterns in order to answer what are the common and task-specific graph properties of networks emerging under a performance-resources trade-off.

3. The setup

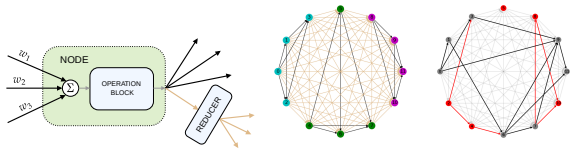
From a DAG to a neural network. We construct the neural network architecture using a fully-connected, directed acyclic graph with 60 nodes, following the approach in (Janik & Nowak, 2020). The nodes correspond to a stack of operations performed on tensors, while the edges define how the information is propagated in the network (see Fig. 1a). Each node performs a block of transformations composed of a weighted aggregated sum of the inputs, followed by a ReLU nonlinearity, Conv-2D mapping and a BatchNorm layer. A residual connection in the form of Conv-1x1 connects the output of the weighted aggregated sum and the output of the whole block. The weights in the aggregation performed in the nodes are given by the weights defined on the edges of the DAG and are trainable as well. Intuitively, each such edge weight corresponds to the portion of the outgoing node's output that is incorporated in the ingoing node's computations.

Following the common approach in image recognition, the whole network is divided into three, equally-sized stages. Within each stage, all nodes perform computations on tensors with the same spatial resolution and number of channels. The first (input) node sets the initial number of channels C for the first stage, keeping the same spatial resolution as the size of the input image. The second and third stages downsample the resolution and increase the number of channels by a factor of 2 and 4, respectively. This transition is performed on every edge that connects different stages with the use of a reduce block as defined in (Janik & Nowak, 2020) – see Fig. 1a and Fig. 1b. Finally, when the computation reaches the last node in the network, a standard global average pooling is applied, followed by a linear transformation to the desired output dimension.

Imposing sparsity. As described above, the connectivity of the neural network is encoded in the weights associated to the edges of the directed acyclic graph. A straightforward way of imposing the change in the network topology is to enforce sparsity on those connections by a L1 loss on the edge weights[1] :

$$L_{resources}(w) = \sum_{e \in edges} |\tanh w_e|. \qquad (2)$$

[1]We use the $\tanh$ activation instead of the *sigmoid* in (Xie et al., 2019; Janik & Nowak, 2020) as we experimentally observe it behaves more stable.



(a) (b) (c)

Figure 1: **(a)** The architecture of a single node in the graph. The black arrows represent connections within a single resolution stage, while the beige ones go to lower resolution stages. **(b)** The architecture of a fully connected DAG. Different node colours represent different stages of computations. **(c)** An example of the thresholding procedure on the DAG from Fig. 1b. The red edges and nodes are the paths that do not contribute to the output and can be further removed.

Since the sign of the final weight does not matter, for the rest of the paper we analyze the absolute value of $\tanh w_e$. As we do not want to *a-priori* bias the architecture we choose a constant initialisation of all edge weights so that $\tanh w_e^{init} = 0.5$.

Pruning the network. As a result of adding $L_{resources}$ to the objective loss, the edge weights are encouraged to have small magnitude. Edges with very low magnitude may be interpreted as paths that do not contribute much to the computation of the network and are therefore redundant and can be safely eliminated.

In order to isolate the active subgraph within the fully connected DAG, we perform straightforward thresholding at the end of the training. Each edge with the absolute value of the weight smaller than a given threshold τ is erased from the graph. Note that such a procedure may lead to paths that do not originate in the input node or do not end in the output node (see Fig. 1c). Those "dead paths" can be further removed from the graph, as they do not contribute to the network output[2].

4. Results

Used Datasets. The main goal of the present paper is to investigate how the learned architectures depend on the difficulty of the classification task. We train the networks on five datasets with increasing *diffculty*: CMNIST, CKMNIST, CFashionMNIST, CIFAR10 and CIFAR10_T8. The CMNIST, CKMNIST and CFashionMNIST from (Janik & Witaszczyk, 2020) are randomly colored versions of MNIST, KMNIST and FashionMNIST datasets, embedded in a 32×32 image space. CIFAR10_T8 (Janik & Witaszczyk, 2020) is a more challenging version of the

[2]This can be implemented by a simple variation of the Kahn's algorithm for topological sort (Kahn, 1962)

standard CIFAR10, where the original images are cut into 8×8 pieces and then shuffled and rotated by multiples of $90°$. The random shuffling and rotations is kept *fixed for all* images in CIFAR10.

We use the same learning regime for all datasets, which is identical to the one used by (Janik & Nowak, 2020) for CIFAR10 (see Appendix A for details). The total loss function is composed of the standard cross-entropy loss to which we add the $L_{resources}$ loss term with coefficient $\lambda = 1e-3$. All models use the same number of initial parameters.

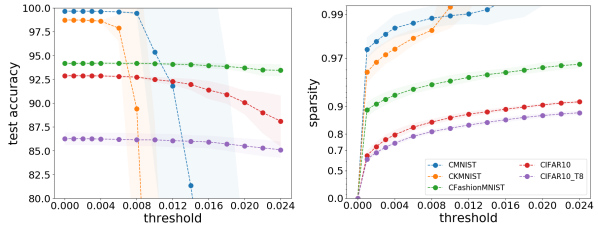


Figure 2: The test accuracy and sparsity plots for given thresholds.

Thresholding. For a set of predefined thresholds τ we compute the mean test accuracy and mean sparsity obtained after downsizing the graph as described in Section 3. The results are presented in Fig. 2. For relatively simple datasets (CMNIST,CKMNIST), the thresholding procedure quickly increases sparsity. Further increase in the sparsity causes the model to deteriorate and finally disconnects the graph. More challenging datasets, as expected, seem to require higher density. For all the datasets erasing edges up to threshold 0.006 practically does not affect the accuracy, while significantly improving the sparsity. This suggests that the deleted links are indeed insignificant and redundant. For the rest of the paper, we investigate the architectures obtained with $\tau = 0.006$ as a reasonable choice for all datasets, which allows for the maximum sparsity without a noticeable drop in performance.[3].

Obtained Networks. In Fig. 3, we show a selection of networks obtained for the various datasets for specific random initialisation of the standard network parameters. Recall, however, that the initial values of all edge weights were set to 0.5 in order not to bias the emerging network connectivity by random initialisation. The graphs are drawn in 2D space using the Kamada-Kawai (Kamada & Kawai, 1989) embedding. We observe visually a steady increase in the network complexity with the difficulty of the classification task. In addition, we see significant structural similarity between the graphs obtained for the same dataset from different random initializations (see Appendix E). In the following we provide a quantitative investigation into these issues.

[3]We emphasise that we do not perform (and do not need) any further fine-tuning. The architectures obtained by us are also capable of achieving high performance when trained again from scratch with arbitrary initialisation - see Appendix B

Table 1: Main structural characteristics of the obtained networks (with threshold 0.006) averaged over 10 different initializations. The initial network has in total 60 nodes, distributed evenly between the three stages and has 863.3k parameters.

	CMNIST	CKMNIST	CFashionMNIST	CIFAR10	CIFAR10_T8
sparsity (all)	0.988 ± 0.003	0.982 ± 0.004	0.935 ± 0.008	0.831 ± 0.016	0.788 ± 0.012
- stage 0	0.981 ± 0.015	0.983 ± 0.009	0.905 ± 0.023	0.791 ± 0.037	0.712 ± 0.034
- stage 1	0.989 ± 0.006	0.989 ± 0.006	0.936 ± 0.028	0.725 ± 0.058	0.754 ± 0.057
- stage 2	0.952 ± 0.020	0.905 ± 0.012	0.822 ± 0.035	0.678 ± 0.046	0.573 ± 0.061
nodes (all)	12.5 ± 2.3	17.0 ± 2.1	26.9 ± 2.3	48.5 ± 2.6	48.2 ± 1.5
- stage 0	3.6 ± 1.8	3.5 ± 1.2	8.8 ± 1.0	15.6 ± 0.7	16.5 ± 1.1
- stage 1	2.9 ± 1.1	3.7 ± 1.3	7.4 ± 1.5	17.3 ± 1.6	14.9 ± 1.3
- stage 2	6.0 ± 1.3	9.8 ± 0.9	10.7 ± 1.5	15.6 ± 1.8	16.8 ± 1.2
parameters	$166k \pm 30k$	$253k \pm 34k$	$333k \pm 34k$	$615k \pm 49k$	$613k \pm 24k$

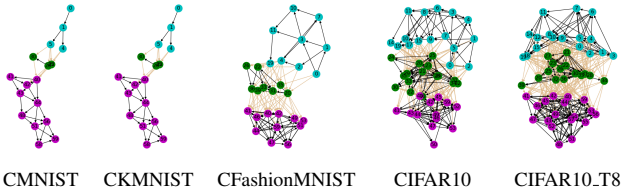


Figure 3: The thresholded networks (at the threshold 0.006) obtained from training for a specific random initialisation. See Appendix E for other initializations.

Structural Characteristics. In Table 1 we have collected the means and standard deviations of the structural characteristics of the obtained thresholded networks ($\tau = 0.006$). These data quantify several important tendencies which can be visually assessed from Fig. 3.

As expected, as the learning task gets easier the networks drop a larger fraction of the original connections. Similarly, the size of the network, as measured by the number of surviving nodes, increases with the complexity of the dataset. The obtained networks are significantly smaller than the original fully connected DAG, yet they do not suffer from a performance loss.

Furthermore, the distribution of surviving computational nodes between the various resolution stages of processing differs systematically between the datasets. Of particular note is the relatively large number of nodes for KMNIST (Kuzushiji-MNIST) in the final stage of processing. It may suggest that the Japanese characters seem to require integration of features at the global level. An opposite tendency is represented by CFashionMNIST, where the whole increase in the number of nodes with respect to KMNIST occurs essentially in the lower resolution stages. It is also interesting to compare from this perspective CIFAR10 and CIFAR10_T8. They differ most in the number of nodes in the middle stage, with the latter dataset having the smaller number. In addition, even though the number of nodes of

stage 0 and stage 2 is similar, there is a significant difference in the number of retained connections within these stages for those two datasets. The high-level stages have lower sparsity so are more interlinked and thus more complex.

Table 2: A selection of graph characteristics of the networks thresholded at 0.006.

dataset	CMNIST	CKMNIST	CFashionMNIST	CIFAR10	CIFAR10_T8
log_paths	4.03 ± 1.03	5.50 ± 0.87	11.76 ± 0.81	18.47 ± 1.31	21.37 ± 1.27
mean_path	7.44 ± 1.05	8.95 ± 0.76	11.28 ± 0.85	16.42 ± 0.99	18.21 ± 1.25
max_path	9.60 ± 1.35	12.00 ± 1.41	18.80 ± 1.75	28.00 ± 2.49	32.60 ± 2.63
ln_communicability	-3.26 ± 1.86	-3.86 ± 1.28	5.52 ± 1.37	11.03 ± 0.90	14.59 ± 0.74
edge_connectivity	1.30 ± 0.48	1.30 ± 0.48	4.80 ± 2.49	13.60 ± 3.44	20.20 ± 1.99
mean_degree	3.48 ± 0.46	3.75 ± 0.30	8.53 ± 0.67	12.31 ± 0.86	15.58 ± 0.73
pca_elongation	0.78 ± 0.15	0.72 ± 0.10	0.52 ± 0.06	0.37 ± 0.06	0.44 ± 0.06

Graph characteristics. In Table 2, we have collected a selection of numerical graph characteristics which differ the most among the obtained networks: the logarithm of the total number of different paths going between the input and output (`log_paths`), the mean and maximal length of such paths (`mean_path`, `max_path`), the `ln_communicability` (Estrada & Hatano, 2008)[4], the `edge_connectivity`, and `pca_elongation` (Janik & Nowak, 2020). See Appendix F for more detailed definitions of those features.

We may observe a steady increase in the first four, path related, features. This captures the increase of complexity of the obtained networks. Note that `mean_path` can be understood as the depth of the network. Similarly, the `edge_connectivity` (minimal number of edges needed to disconnect the graph) can be understood as a proxy for the breadth of the graph or the extent of interconnections. Its increase for the more challenging datasets is clear.

The last observable in Table 2, `pca_elongation` was used in (Janik & Nowak, 2020) to define the *quasi-1-dimensional* (Q1D) family — a class of well-performing architectures that was identified by analyzing static connectives based on radnom DAGs. A graph was said to satisfy the Q1D criterion, if `pca_elongation`> 0.25 and `edge_connectivity`> 1. It is worth noting that the vast majority of the graphs obtained in the learning process have significantly larger `pca_elongation` than the 0.25 value appearing in the definition of Q1D (see Table 2). It can also be observed in Fig. 3 that the networks for CMNIST, CKMNIST and CFashionMNIST have clearly a quasi-1-dimensional structure[5]. One should emphasise that the initial network architecture, the fully connected DAG, has `pca_elongation`$= 0$, which means that the Q1D structure emerges in the process of learning.

[4](the logarithm of) the communicability – an element of the exponent of the adjacency matrix which measures a weighted sum over walks between the input and output node

[5]Q1D is sometimes violated for the simplest networks which can be split by cutting a single edge.

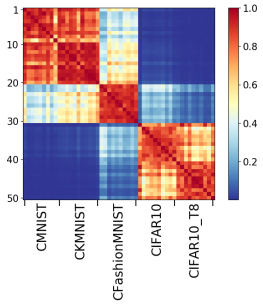


Figure 4: Network similarity matrix (see Fig. 9 in Appendix E).

In order to visualise to what extent the obtained networks for a given dataset are similar between themselves and different from networks for other datasets, for each of the 50 obtained networks we have collected the features exhibited in Tables 1 and 2 (apart from the number of parameters), standardised them and evaluated pairwise similarity using the standard RBF kernel. The result is shown in Fig. 4. One may observe that for the same dataset, all the obtained graphs are strongly correlated. Moreover, networks working on tasks of comparable difficulty also seem to share similarities like CMNIST and CKMNIST, or CIFAR10 and CIFAR10_T8. This confirms that the uncovered wiring topology is firmly related to the problem it is solving.

Conclusions. In the present paper we have analyzed networks obtained in a simple but effective method of learning the network's size and topology under a performance-resources trade-off. Basing the construction on an initial fully connected DAG, allows the network to have full freedom of exploring local and global connectivity.

The obtained networks are clearly correlated with the difficulty of the dataset. On the one hand, we observe certain common features like the more involved connectivity of the high-level processing and a Q1D structure. On the other hand, we see differences specific to particular datasets like the proportion of nodes and connections in various resolution stages. The fact that the obtained graphs are characteristic to the specific learning tasks opens up a fascinating possibility of understanding them as representing hidden internal structures of the particular datasets, via a mapping:

$$\mathcal{D} \longrightarrow P(\mathcal{G}) \tag{3}$$

which associates to each dataset $\mathcal{D}$ (more precisely learning task) a distribution of graphs. Moreover this probability distribution seems to be quite localised, as we have observed in Fig. 4 that the obtained networks have consistently characteristic features for a given dataset. It would be very interesting to explore the properties of those distributions further.

References

Anwar, S., Hwang, K., and Sung, W. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):1–18, 2017.

Baker, B., Gupta, O., Naik, N., and Raskar, R. Designing neural network architectures using reinforcement learning. In *International Conference on Learning Representations*, 2016.

Blalock, D., Ortiz, J. J. G., Frankle, J., and Guttag, J. What is the state of neural network pruning? *arXiv preprint arXiv:2003.03033*, 2020.

Cheng, Y., Wang, D., Zhou, P., and Zhang, T. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

Elsken, T., Metzen, J. H., and Hutter, F. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

Estrada, E. and Hatano, N. Communicability in complex networks. *Physical Review E*, 77(3):036111, 2008.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pp. 1135–1143, 2015.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Janik, R. A. and Nowak, A. Neural networks on random graphs. *arXiv preprint arXiv:2002.08104*, 2020.

Janik, R. A. and Witaszczyk, P. Complexity for deep neural networks and other characteristics of deep feature representations. *arXiv preprint arXiv:2006.04791*, 2020.

Kahn, A. B. Topological sorting of large networks. *Communications of the ACM*, 5(11):558–562, 1962.

Kamada, T. and Kawai, S. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1): 7–15, 1989.

LeCun, Y., Denker, J. S., and Solla, S. A. Optimal brain damage. In *Advances in neural information processing systems*, pp. 598–605, 1990.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Lin, J., Rao, Y., Lu, J., and Zhou, J. Runtime neural pruning. In *Advances in Neural Information Processing Systems*, pp. 2181–2191, 2017.

Liu, H., Simonyan, K., and Yang, Y. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018a.

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018b.

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pp. 4780–4789, 2019.

Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*, pp. 2074–2082, 2016.

Xie, S., Kirillov, A., Girshick, R., and He, K. Exploring randomly wired neural networks for image recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1284–1293, 2019.

Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., and Hutter, F. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pp. 7105–7114, 2019.

Zhang, X., Huang, Z., and Wang, N. You only search once: Single shot neural architecture search via direct sparse optimization. *arXiv preprint arXiv:1811.01567*, 2018.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

A. Training Regime

We train the fully connected DAG architectures 10 times for each dataset using different initialisation but the same learning regime, adopted from (Janik & Nowak, 2020) for CIFAR10. All models are trained for 100 epochs using the SGD algorithm with starting learning rate 0.1, momentum 0.9, batch size 128 and weight decay 1e-4. In the 80th and 90th epoch the learning rate is decreased by factor 10. The initial number of channels is set to $C = 11$ in order for the networks to have approximately the same number of parameters as ResNet-56 (He et al., 2016). The models are trained and evaluated on the standard train/test splits defined for each dataset. For CIFAR10_T8 we adopt the same split as for CIFAR10. The objective loss function is composed of the standard cross-entropy loss to which we add the $L_{sparsity}$ loss term with coefficient $\lambda = 1e - 3$.

For the retrain experiments we use the same settings but with different initialisation seed. All networks were trained using the GeForce RTX 2080 Ti graphic card.

B. Training with Reintialization

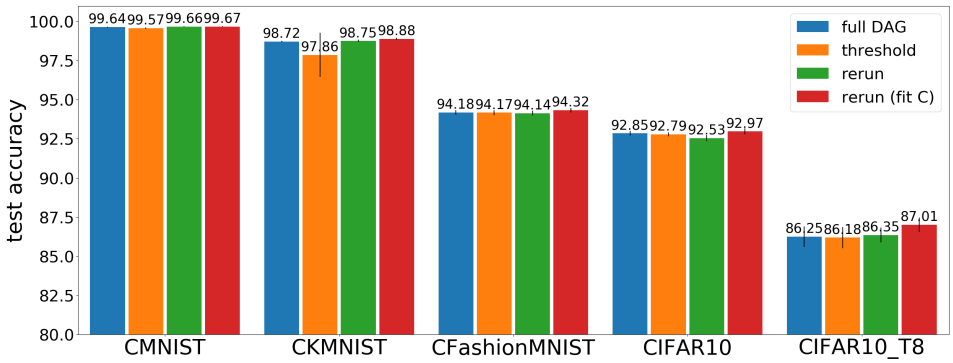


Figure 5: The test accuracy of the initial fully-connected DAG, the thresholded model, the thresholded model after retraining from scratch, and same but with C increased to match the number of parameters of ResNet-56.

We validate whether the obtained architectures are also capable of achieving high performance when trained again from scratch with arbitrary initialisation. We consider two scenarios. In the first one we train the thresholded network from scratch using different initialization. In the second setup we additionally fit the number of initial channels C separately for each of the thresholded networks, so that the number of parameters approximately matches the number of parameters of ResNet-56 (and the original, fully connected DAG network).

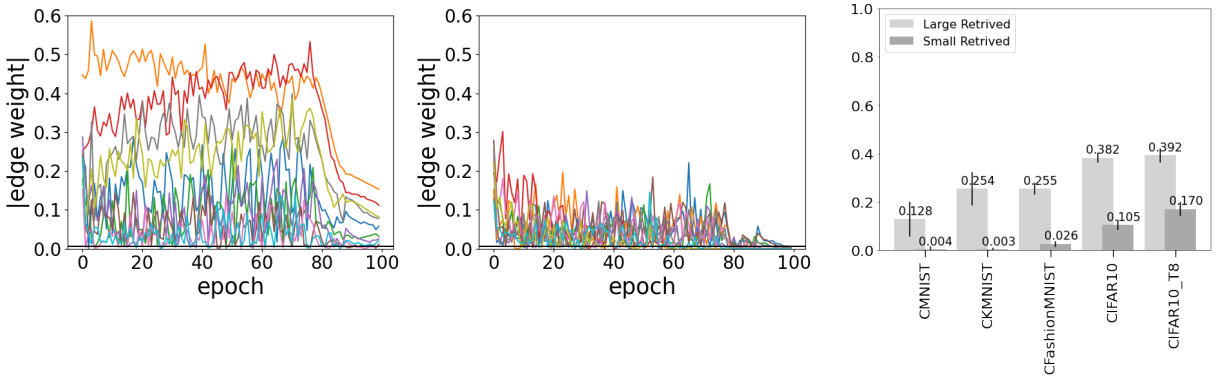
As observed in Fig. 5, retraining the obtained architectures with the same learning regime yields comparable or better performance to the one achieved by the fully connected DAG. This shows that the resulting connectivity patterns are indeed meaningful and not limited to a specific weight initialisation. Moreover, in both retraining scenarios, the time of one epoch is significantly smaller than for the complete DAG - see Section C in the Appendix.

C. Training Time

In Table 3 we report the mean time of one epoch during the training of the fully connected DAG network and the retrained networks obtained for threshold 0.006. In the latter case, as described in the paper, we consider two situations: retraining with the same number of initial channels $C = 11$ as the original, complete DAG and fitting the number of channels C separately for each architecture in order for the networks to have approximately the same number of *parameters* as the complete DAG. All models were trained using the GeForce RTX 2080 Ti graphic card. Although for different computational infrastructure the values may vary a bit, the advantage gained after pruning is indisputable.

Table 3: The mean epoch time during the training of the original graph, the obtained network with number of initial channels C being set to the same value as the fully connected DAG, and the obtained network when C was fitted separately for each architecture.

dataset	full DAG	rerun	rerun (fit C)
CMNIST	151.99±17.73	21.65±3.28	21.81±3.60
CKMNIST	147.03±19.55	26.05±4.39	25.84±7.00
CFashionMNIST	149.71±19.46	41.09±16.92	32.77±2.33
CIFAR10	120.98±16.59	75.29±3.07	59.23±13.23
CIFAR10_T8	107.42±5.53	76.09±3.63	69.45±5.48



(a) (b)

Figure 6: **(a)** The magnitudes of a selection of the retained (left) and eliminated (right) edges during the training. **(b)** The mean DSC score between the set of edges in the thresholded graph and the set of the same size composed of edges with the largest and smallest weights after the first training epoch.

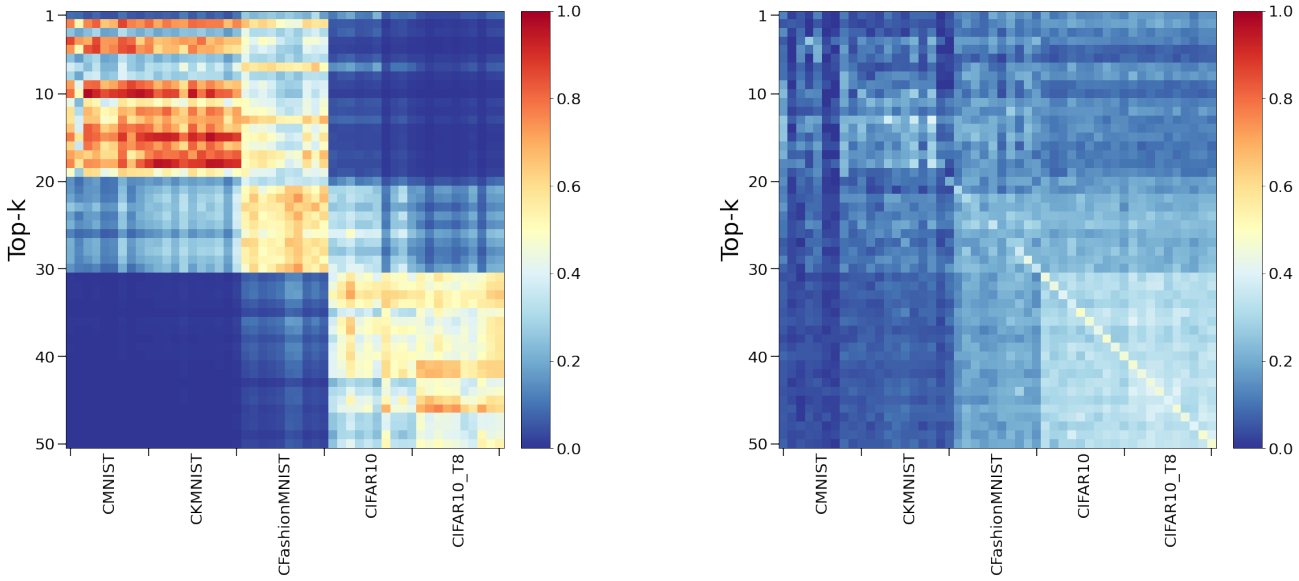
D. Edge Weight Dynamics

In this section we would like to comment on the behaviour of the magnitudes of the edge weights during training. In Fig. 6a we show the evolution of a subset of weights which get retained after thresholding and of those which get eliminated. In addition, in Fig. 6b, we compute the DSC score[6] between the set of edges in the thresholded graph and the set of the same size composed of edges with the largest and the smallest weights after the first training epoch. We observe that weights which are initially very large typically stay large throughout the training (see left plot in Fig. 6a). However, as observed in Fig. 6b, such connections are not the majority among the retained ones, accounting for at most circa 40% of all final edges (for CIFAR10_T8). In addition, weights which are small in the first epoch of the training tend to be eliminated for simple datasets such as CMNIST and CKMNIST, however are still subject to change for harder datasets such as CIFAR10 and CIFAR10_T8. This is intuitive, as easier tasks require less training iterations to converge and hence the distribution of the weights in the first stage of the training is more similar to the distribution obtained at the end of optimization.

Finally, we would like to point out that not all of the removed edges have constantly the smallest magnitudes during the training. This can be observed in the right plot of Fig. 6a. The weights fluctuate, mostly maintaining values larger than the used threshold (and comparable to the magnitudes of many of the retained edges), and eventually settle down to the final values in the last stage of training, after the reduction of the learning rate. We hypothesize that this can be interpreted as exploring various wiring patterns, before picking out a particular connectivity at the end of the optimization.

In addition, for each trained network we store the values of the edge weights at the end of each epoch. We analyze the changes in the magnitudes of the retained and eliminated weights in Table 4. It may be observed that typically at least half

[6] Sørensen–Dice coefficient between edge sets E_a and E_b : $DSC_{E_a,E_b} = |E_a \cap E_b|/(|E_a| + |E_b|)$



(a) (b)

Figure 7: The structural similarity **(a)** and the DSC similarity **(b)** between the thresholded networks and networks composed of the largest edges of the first epoch. For simple MNIST-like datasets the strctural similarity is quite large, at the same time being much weaker for the more challenging tasks. Nevertheless, the considered edge sets are much different, as measured by the DSC similarity. Note that the vertical lines of low structural similarity for CMNIST datasets (in the top right corrner of plot **(a)**) are caused by the weakly-connected condition. Those graphs would be disconnected if one would not include more high magnitude edges.

of the retained after thresholding edges had values lower than the target threshold. A single retained weight experience such change on average at least circa three times for the simple datasets and five to six times for the harder datasets. This shows that the sole fact of achieving a very small weight value at some step of the optimization process does not prevent the edge from recovering at a later point. In addition, at least half of the eliminated edges after firstly becoming smaller than the target threshold experience values significantly larger than the target threshold, which is coherent with the fluctuations of the weights in the right plot of Fig. 6a.

Structural and Edge Similarity In addition to the discussion above we further analyze the networks obtained by selecting the edges with the largest magnitude in the first epoch of training to the networks obtained by the thresholding framework in terms of graph properties.

To this end we construct graphs which are composed of the set of k edges which have the largest absolute value of weights in the first epoch of training. We pick the value k in such a manner that the resulting edge set is the smallest edges set that renders the graph weakly-connected and its size it at least as large as the size induced by the mean sparsity. We refer to these topologies as "top-k" graphs. Note that we need the weakly-connected condition since for simple datasets such as CMNIST and CKMNIST the networks obtained by simply selecting the set of the size given by the mean sparsity are often disconnected. This would pose computational problems in many graph properties from Table 2 and prevent us from computing the RBF kernel similarity.

Next, as before, for all datasets we compute the structural (RBF) and DSC similarity between all of the thresholded networks and the corresponding top-k graphs. The results are presented in Fig. 7. The structural similarity is strong for the simpler datasets and much weaker for the more complex datasets. This is intuitive, since easier tasks require less training iterations to converge, and thus the structure obtained in the first epoch of training may be already similar to the one achieved at the end. However, note that the obtained edge sets are much different, as measured by the DSC score in Fig. 7b. This shows that the the graphs obtained at the end of the thresholding framework are indeed distinct from the graphs arising in the early stage of training.

Table 4: The table summarizes some properties of the edge evolution during training for different datasets. (A) reports the percentage of the retained edges which at least for one epoch were below the target threshold during the training. (B) reports the mean number of epochs at the end of which the retained edges had values lower than the target threshold. (C) reports the percentage of the eliminated edges which after firstly becoming smaller than the target threshold experienced values *ten times* larger than the target threshold. (D) reports the mean number of epochs at the end of which the eliminated edges experienced values larger than the target threshold after firstly becoming smaller than the target threshold. The last column presents the mean number of edges in the thresholded network.

dataset	A	B	C	D	number of edges
CMNIST	50.00 ± 6.50	3.79 ± 1.76	49.33 ± 266.80	1.64 ± 0.57	23.00 ± 5.57
CKMNIST	48.37 ± 7.14	3.82 ± 1.79	52.76 ± 43.24	2.19 ± 0.16	33.70 ± 6.15
CFashionMNITS	61.95 ± 13.32	5.78 ± 0.72	47.44 ± 101.95	1.90 ± 0.24	115.10 ± 13.57
CIFAR10	53.75 ± 23.85	5.34 ± 0.59	74.93 ± 23.98	5.93 ± 0.43	299.00 ± 26.51
CIFAR10_T8	63.41 ± 17.51	6.61 ± 0.33	76.08 ± 25.40	5.38 ± 0.31	375.50 ± 20.12

Table 5: The test accuracy obtained for the standard dense model, the trained model after thresholding (with threshold equal to 0.006) and the thresholded model retrained from scratch.

	standard	thresholded	retrained (fit C)
Full DAG	92.849 ± 0.152	92.786 ± 0.123	92.967 ± 0.253
Subgraph	92.901 ± 0.193	92.886 ± 0.220	93.265 ± 0.178

D.1. Introducing edge bias.

Implying sparsity by magnitude based pruning and L1 regularization allows us to investigate the impact of introducing structural bias in the networks, which can be easily implemented by altering the underlying network connectivity. Instead of performing the training on the fully-connected DAG one may decide to remove some connections beforehand, performing the optimization on a subgraph. Since local edge connectivity is typically believed to lead to better performing models (Janik & Nowak, 2020; Elsken et al., 2019), we consider a graph in which all edges connecting nodes i and j for which $|i-j| > 10$ are removed. The results are presented in Table 5. The standard test accuracy and the accuracy after thresholding are quite similar in the Full DAG and subgraph approaches. However, in the latter case, the performance of the obtained models when trained from scratch is slightly better and has a lower standard deviation.

E. Networks Obtained for Various Datasets

In Fig. 8 we present the UMAP embedding of the obtained architectures for thresholds equal or smaller than 0.006. The embedding was computed on the standardised features from Tabels 1 and 2 from the paper (apart from the number of parameters). It is evident that networks solving the same task tend to cluster, being similar to each other. This similarity can be also visually observed in Fig. 9, where we demonstrate a subset of the obtained networks for threshold 0.006 for different datasets with various initialisations.

F. Graph Characteristics

In table 6 we provide a more comprehensive description of the graph properties which were used in the main paper. The names with asterisk (*) indicate that a given value was computed with the use of the `networkx` package[7]

[7] https://networkx.org/

Table 6: The summary of the used graph properties. The names with asterisk (*) indicate that a given value was computed with the use of the `networkx` package.

property	**description**
`log_paths`	The logarithm of the number of all paths that start in the input node and and in the output node.
`mean_path`	The mean length of a path in the graph.
`max_path*`	The maximum path length in the graph.
`ln_communicability*`	The logarithm of the communicability between the input and the output node. The communicability of a pair of nodes (u, v) in a graph $G = (V, E)$ is the sum of closed walks of different lengths starting at node u and ending at node v: $$C(u,v) = \sum_{i}^{n} \phi_u^i \phi_v^i e^{\lambda_i},$$ where ϕ_j^i and λ_i are the j-th value in the i-th eigenvector and the i-th eigenvalue of the spectral decomposition of the adjacency matrix.
`edge_connectivity*`	The edge connectivity between the input and the output node. The edge connectivity between a pair of nodes (u, v) is the minimum number of edges that need to be removed in order to disconnect u form v (two nodes are considered disconnected if there is no path between them).
`mean_degree*`	The degree of a node in a graph is the number of edges going from our to that node. The mean degree is the mean degree over all nodes.
`pca_elongation`	The PCA elongation is computed in relation to a graph embedding. First, the graph nodes are embedded into a two-dimensional space using the Kamada-Kawai algorithm. Next, a PCA decomposition is performed on the obtained embedding. PCA elongation is then the defined as: $$pca\_elongation = 2V_1 - 0.5,$$ where V_1 is the variance ratio corresponding to the largest eigenvalue computed during the decomposition. Note that this as a non-trivial characteristic of the graph, since the Kamada-Kawai algorithm encodes the internal information related to the graph structure. Large PCA elongation are typically associated with locally connected networks that have a sequential global connectivity.

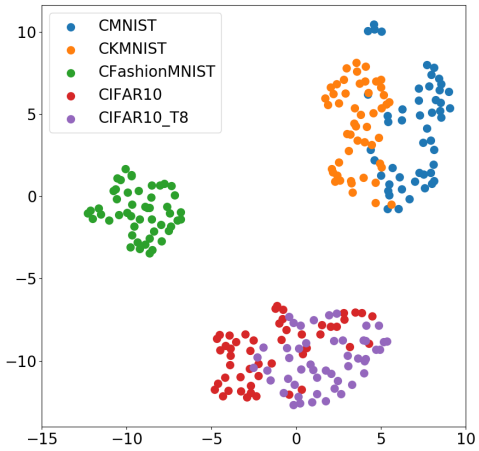


Figure 8: The UMAP embedding of the networks obtained for thresholds $\tau \leq 0.006$. Different colours represent different datasets.

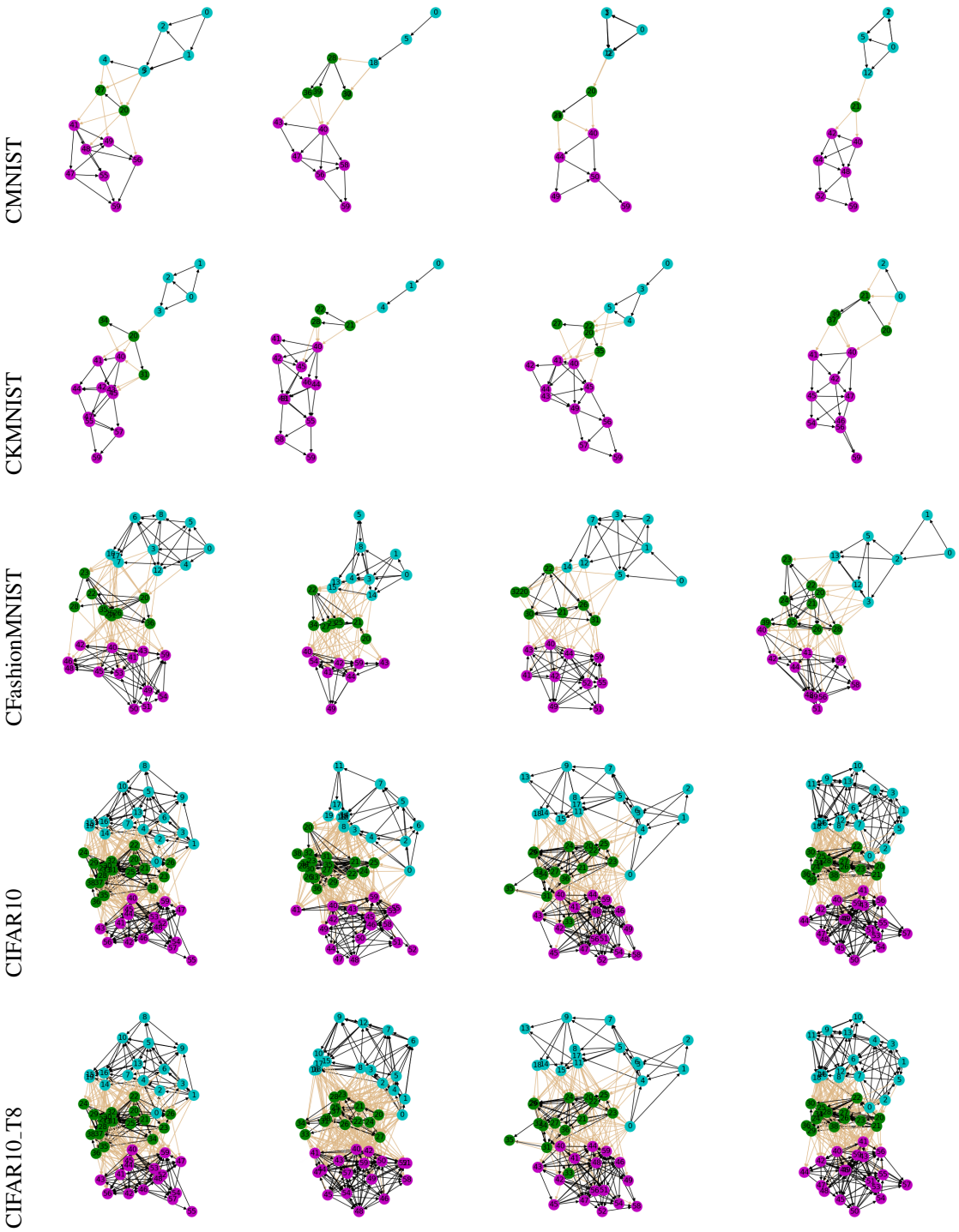


Figure 9: The thresholded networks (at the threshold 0.006) obtained from training with several random initialisations.