SWANN: Small-World Architecture for Fast Convergence of Neural Networks

Mojan Javaheripi, *Graduate Student Member, IEEE*, Bita Darvish Rouhani, *Student Member, IEEE*, and Farinaz Koushanfar, *Fellow, IEEE*

***Abstract*— On-device intelligence has become increasingly widespread in the modern smart application landscape. A standing challenge for the applicability of on-device intelligence is the excessively high computation cost of training highly accurate Deep Learning (DL) models. These models require a large number of training iterations to reach a high convergence accuracy, hindering their applicability to resource-constrained embedded devices. This paper proposes a novel transformation which changes the topology of the DL architecture to reach an optimal cross-layer connectivity. This, in turn, significantly reduces the number of training iterations required for reaching a target accuracy. Our transformation leverages the important observation that for a set level of accuracy, convergence is fastest when network topology reaches the boundary of a Small-World Network. Small-world graphs are known to possess a specific connectivity structure that enables enhanced signal propagation among nodes. Our small-world models, called SWANNs, provide several intriguing benefits: they facilitate data (gradient) flow within the network, enable feature-map reuse by adding long-range connections and accommodate various network architectures/datasets. Compared to densely connected networks (e.g., DenseNets), SWANNs require a substantially fewer number of training parameters while maintaining a similar level of classification accuracy. We evaluate our networks on various DL model architectures and image classification datasets, namely, MNIST, CIFAR10, CIFAR100, and ImageNet. Our experiments demonstrate an average of ≈ 2.1× improvement in convergence speed to the desired accuracy.**

***Index Terms*— Deep learning, on-device training, small-world networks.**

I. Introduction

DEEP learning models are increasingly popular for various automated learning tasks, particularly in visual computing applications. Recently, there has been a shift to incorporate DL training and execution on smart devices rather than offloading the computations to cloud-based servers. This transition is motivated by the compelling properties of on-device computation, e.g., preserving user data privacy and eliminating the need for continuous network connection. A standing challenge for on-device intelligence is the limited resources available on embedded devices that slow down DL execution compared to the cloud. The constraints of the embedded environment are specially critical for lengthy DL training. Contemporary DL models require high number of training iterations to converge, hindering their applicability to on-device *learning*. In this paper, we focus on reducing the required training iterations for convergence, thereby paving the way for on-device learning applications such as federated learning and (local) personalization.

The literature in Neural Architecture Search (NAS) is primarily focused on generating compact and accurate Deep Neural Networks (DNNs) for inference. To increase the search flexibility and reach a higher accuracy, a recent body of work in NAS explores the use of irregular wirings, aka bypass connections [1]–[4]. These bypasses connect (non-consecutive) layers in the DL architecture that would otherwise be disconnected in a traditional DNN. While prior work in NAS can reduce the computational complexity of DNN inference, there has been little focus on the training cost of the obtained DNNs for reaching their target accuracy. To enable on-device learning, we study irregular network wirings through the lens of DL training speed.

We propose a novel methodology that transforms the topology of conventional DNNs such that they reach an optimal cross-layer connectivity. This, in turn, significantly reduces the number of training iterations required for reaching a target accuracy. This transformation is based on our observation that the pertinent connectivity pattern highly impacts training speed and convergence. To ensure computational efficiency, our architectural modification takes place **prior** to training. Thus, the incorporated connectivity measure must be independent of network gradients/loss and training data. Towards this goal, we view DNNs as graphs and revisit Small-World Networks (SWNs) [5] from graph theory to transform DNNs into highly-connected small-world topologies. Watts-Strogatz SWNs [5] are widely used in the analysis of complex graphs; Due to SWNs' specific connection pattern, these structures provide theoretical guarantees for considerably decreased consensus times [6]–[8].

Our network modification algorithm takes as input a conventional DNN architecture and enforces the small-world property on its topology to generate a new network, called SWANN. We leverage a quantitative metric for *small-worldness* and devise a customized rewiring algorithm. Our algorithm restructures the inter-layer connections in the input DNN to find a

Manuscript received May 15, 2021; revised September 8, 2021 and October 16, 2021; accepted October 18, 2021. Date of publication November 4, 2021; date of current version December 13, 2021. This article was recommended by Guest Editor T. Mohsenin. *(Corresponding author: Mojan Javaheripi.)*

Mojan Javaheripi and Farinaz Koushanfar are with the Department of Electrical and Computer Engineering, University of California San Diego, San Diego, CA 92093 USA (e-mail: mojan@ucsd.edu).

Bita Darvish Rouhani is with Microsoft, Redmond, WA 98052 USA.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JETCAS.2021.3125309.

Digital Object Identifier 10.1109/JETCAS.2021.3125309

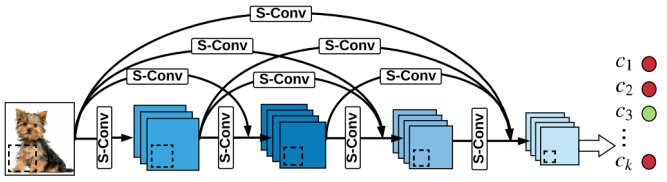

Fig. 1. Schematic representation of the connections within a small-world DNN. An arbitrary neuron's output is connected to selected neurons in the proceeding layers via sparse connections (convolutions) denoted by *S-CONV*.

topology that balances *regularity* and *randomness*, which is the key characteristic of SWNs [5]. Small-world property in DNNs translates to an architecture where all layers are interlinked via sparse connections, an example of which is shown in Fig. 1.

SWANNs have similar quality of prediction and number of trainable parameters as their baseline feed-forward architectures, but due to the added sparse links and the optimal SWN connectivity, they warrant better data flow. In summary, our architecture modification has three main properties: (i) It removes non-critical connections. (ii) It increases the degrees of freedom during training, allowing faster convergence. (iii) It provides customized data paths in the model for better cross-layer information propagation.

We conduct comprehensive experiments on various network architectures and showcase SWANNs' performance on popular image classification benchmarks including MNIST, CIFAR10, CIFAR100, and ImageNet. Our small-world DNNs achieve an average of 2.1-fold reduction in training iterations required to achieve comparable test accuracy as the baseline models. We further compare SWANN with the DenseNet model and show that with 10× fewer parameters, SWANNs demonstrate identical performance during training. Finally, as a popular application of on-device learning, we benchmark SWANN in the federated learning scenario where multiple embedded devices collaboratively train a global model on their local datasets. In the federated scenario, SWANN reduces the number of (global) training iterations by 1.4× on average, thereby reducing both the computation and communication in decentralized learning.

II. Related Work and Background

A. Related Work

A line of research has focused on the addition of bypass connections to the DNN architecture to enhance inter-layer information flow and enable feature reuse. Perhaps the pioneer work is ResNet [9], which uses identity links (skip connections) to connect non-sequential layers. ResNet's skip connections follow a modular structure which results in redundancies since not all identity links are necessary as shown by [10]. DenseNets [1] are another example that use skip connections to connect each layer to all its preceding layers in a block. This is done by concatenating previous layers' feature-maps and using them as the input. Another work [11] argues that such dense connectivity incurs redundancies since earlier features might not be required in later layers. The authors of [11] prune the redundancies to generate a more efficient architecture for the DNN inference phase.

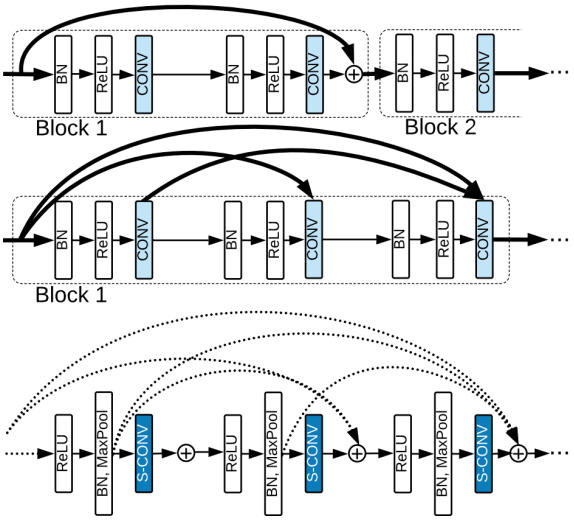

Fig. 2. Information flow within a ResNet (top), DenseNet (middle), and SWANN (bottom) network. Here, *CONV*, *BN*, *ReLU* denote a convolution kernel, batch normalization, and non-linear activation, respectively, and our customized sparse convolutions are shown as *S-CONV*. Normal inter-layer connections are represented with bold lines and dotted lines are SWANNs' selective inter-links.

Fig. 2 illustrates the connection pattern in a ResNet, DenseNet, and SWANN architecture. In contrast to these two models, SWANN is not structured upon fixed building blocks and therefore can adapt to any given network architecture. Different from DenseNets which only accommodate fully dense connections, SWANNs leverage customized sparse convolutions. This sparsity enables selective connectivity between pairs of layers that enhance convergence speed while ensuring a low redundancy. Using sparse connections is explored in [12] where a trained DNN is pruned in a post-processing phase to reduce parameter count and improve inference performance. However, the pruning does not directly incorporate small-world characteristics and there is no analysis to show that the pruned networks are small-world. Additionally, the focus of [12] is on the inference phase and the pruning is performed after the DNN is trained. Rather, our approach is performed prior to DNN training with the goal of improving convergence. Finally, since the number of parameters is reduced in [12], the accuracy degrades compared to the baseline DNN. SWANN keeps the total number of parameters constant when converting the DNN to a small-world, thereby maintaining the accuracy.

Recent literature in NAS suggests exploration of irregularly wired DNN topologies [13], [14] and random connections [2]–[4] to obtain higher accuracies. Irregularly wired DNNs deviate from the regular DNN topology where the output of each layer is only fed to its immediately preceding layer. From the accuracy and computation perspective, the irregularly wired models have been shown to outperform regular DNNs for inference. The accuracy gains of random DNN connections has also been recently explored from a theoretical standpoint in [15]. Prior work also explores customized compilers and scheduling schemes for the irregularly wired networks to boost their execution performance on edge devices [16].

Perhaps the first investigation of SWNs in the context of machine learning was performed in [17], where small-scale

Multilayer Perceptrons (MLPs) are transformed to a small-world graph. The paper shows that the small-world graph achieve lower error after the same total number of training iterations. Similarly, [18] transforms simple MLPs to SWN graphs and study the accuracy benefits for diagnosis of diabetes. SWANN substantially differs from these works as our solution is applicable to contemporary convolutional neural networks containing various linear kernels and irregular long/short-range connections. Additionally, [17], [18] use a different mathematical model and metric for small-worldness. Authors of [19] randomly rewire MLPs to improve their function approximation capabilities. While the models are generated using random rewiring, there is no systematic choice of rewiring probability and no analysis of small-world characteristics for the developed models. Follow-up work studied the properties of randomly rewired DNNs [2], [4], [15] on classification accuracy, however, they did not specifically focus on small-world characteristics.

In summary, prior work mainly focuses on accuracy gains of long-range connections with little attention to the training process. To the best of our knowledge, SWANN is the first work to intertwine the small-world property with DNNs and examine the acquired networks in terms of both training convergence speed and accuracy.

B. Background: Small-World Networks

Watts and Strogatz [5] observed that real-world complex networks, e.g., the anatomical connections in the brain and the neural network of animals, cannot be modeled using existing regular or random graph classes. As such, they introduced the new category of *small-world networks*. Members of the small-world class have two main characteristics: 1) They have a small average pairwise-distance between graph nodes. 2) Nodes within the graph exhibit a relatively high (local) clustered structure. The first property is mainly associated with random graphs while the second property is prominent in regular graphs. SWNs have shown significantly enhanced signal propagation speed, consensus, synchronization, and computational capability [8], [20]–[23].

Randomness is introduced into a regular graph structure by iterative removal and addition of edges with probability, p, in order to construct an SWN. Fig. 3 demonstrates the transition between a regular graph and the corresponding random graph as the rewiring probability increases from 0 to 1. Intermediate values of p interpolate between complete regularity and randomness to generate an SWN.

III. SWANN: Small-World DNNs

We propose to restructure the inter-layer connections in a DL model such that its topology falls into the small-world category while the total number of parameters in the network is held constant. Throughout the paper, we use the terms DL model and DNN interchangeably but emphasize that our approach is easily applicable to models without convolution layers, e.g., Multi-Layer Perceptrons (MLPs).

In the following, we first elaborate on the small-world criteria and introduce methods to distinguish SWNs from other topologies (Sec. III-A). We then explain our conversion of an arbitrary DNN into its equivalent SWANN (Sec. III-B). Lastly, we delineate our implementation and formalize the operations performed in an SWANN (Sec. III-C).

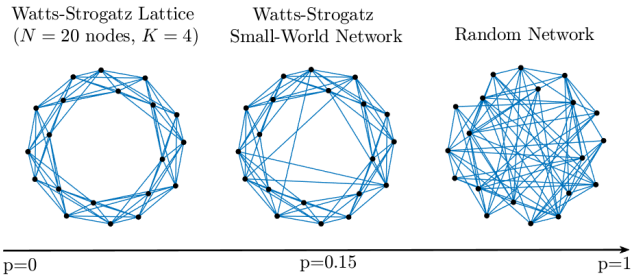

Fig. 3. Transition of a regular graph to a completely random network. Intermediate values of the random rewiring probability, p, generate SWNs, i.e., clustered structures where any arbitrary node pair is connected by a few edges.

A. Metric for Small-Worldness

To examine the small-world property for a given graph, we study two properties, namely, the *characteristic path length* (L) and the *global clustering coefficient* (C). For a given graph, L is calculated by taking the average of minimum path lengths over all node pairs. In this context, the minimum path length is equal to the smallest number of edges one must traverse to get from the first node to the second, or vice versa. The clustering coefficient C is a measure for the connection density between neighbors of any node in the network and is formulated as follows:

$$C = \sum_{i=1}^{V} C_i, \quad \text{where } C_i = \frac{E_i}{\frac{1}{2}N_i(N_i - 1)} \tag{1}$$

Here, C_i denotes the local clustering coefficient of the i^{th} node (v_i), E_i is the number of edges between neighbors of v_i, N_i is the number of neighbors of v_i, and V is the total number of nodes. As shown, the global clustering coefficient, C, is the mean of local coefficients. Regular lattices are highly clustered (large C) but have a very high L which conflicts with our desire to create bypass connections in the DNN. A completely random graph enjoys a small L but lacks clustering. Small-world graphs strike a balance between randomness and regularity by having a large C and small L.

By definition, a graph is small-world if it has a similar L but higher C compared to an $Erd\ddot{o}s - Re'nyi$ ($E - -R$) random graph [24] constructed using the same number of nodes and edges. Let us denote the clustering coefficient and the characteristic path length of a given graph (G) by C_G and L_G, respectively. In a similar fashion, we represent the corresponding characteristics of the equivalent $E--R$ random graph by C_{rand}, L_{rand}. We use a quantitative measure of the small-world property form [25] which categorizes a network as an SWN if $S_G > 1$ where S_G is calculated using Eq. (2).

$$S_G = \frac{\gamma_G}{\lambda_G}, \quad \gamma_G = \frac{C_G}{C_{rand}}, \quad \lambda_G = \frac{L_G}{L_{rand}} \tag{2}$$

B. Acquiring the Small-World Architecture

1) Graph Generation: In order to modify a given DNN architecture and generate the equivalent SWANN, we first model all connections within the network as a graph representation. In this context, a connection is defined as a linear operation performed between an input element and a trainable weight (network parameter) found in *Convolution* ($CONV$) and *Fully-Connected* (FC) layers. For $CONV$ layers, each feature-map channel is represented by one node and each edge represents a $k \times k$ kernel. For FC layers each neuron is assigned a separate node and the edges correspond to weight matrix elements.

2) Architecture Search: After generating the graph that corresponds to the input DNN architecture, we proceed to find the equivalent SWANN. To perform this task, the initial graph is randomly rewired with different probabilities, $p \in [0, 1]$, similar to Fig. 3. For each rewired graph, we compute the characteristic path length L and clustering coefficient C and use the captured pattern for each criterion to detect the small-world topology using the small-worldness measure in Eq. (2).

a) Rewiring policy: Let us denote an edge with $e(v_i, v_j)$ where v_i and v_j are the start and end nodes. To perform random rewiring with probability p, we visit all edges in the graph once. Each edge is rewired with probability p or kept the same with probability $1 - p$. If the edge needs to be rewired, a new second node $v_{j'}$ is randomly sampled from the set of nodes that are non-neighbor to the edge's start node, v_i. This second node is selected such that no self-loops or repeated links exist in the rewired graph. Once the destination node is chosen, the initial edge, $e(v_i, v_j)$ is removed and replaced by $e(v_i, v_{j'})$. Algorithm 1 summarizes the rewiring procedure performed on a baseline DNN to generate the rewired counterpart. Here, $\mathcal{N}$ is the total number of layers in the network, V_l denotes the nodes in the l^{th} layer, and $E_{l,l+1}$ is the set of edges connecting neurons in layer l to its preceding layer $(l + 1)$. Input layer is shown as $l = 0$.

Algorithm 1 Random Rewiring Procedure

Input: input DNN's graph G, rewiring probability p
Output: rewired network G_{rwd}
$G_{rwd} \leftarrow G$
for $l = 0$ **to** $(\mathcal{N} - 2)$ **do**
  for v_i **in** V_l **do**
    for v_j **in** V_{l+1} **do**
      if $|E_{l,l+1}| \geq 1$ **and** $e(v_i, v_j) \in G_{rwd}$ **then**
        $r \sim \mathcal{U}[0, 1]$
        if $r \leq p$ **then**
          $G_{rwd} \leftarrow \{G_{rwd} - e(v_i, v_j)\}$
          while $e(v_i, v_{j'}) \in G_{rwd}$ **do**
            $v_{j'} \sim \{V_{l+2} \cup \cdots \cup V_{\mathcal{N}}\}$
          $G_{rwd} \leftarrow \{G_{rwd} + e(v_i, v_{j'})\}$

Fig. 4 demonstrates the removal/addition of edges in the DNN architecture during our rewiring procedure. Note that our rewiring methodology does not alter the number of connections in the DNN. As a result, the total number of trainable parameters in the final obtained SWANN equals that of the original network.

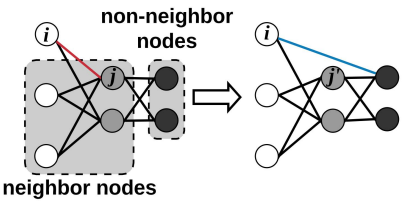

Fig. 4. Our proposed rewiring algorithm replaces edges to the subsequent layer (red) with long-range edges (blue).

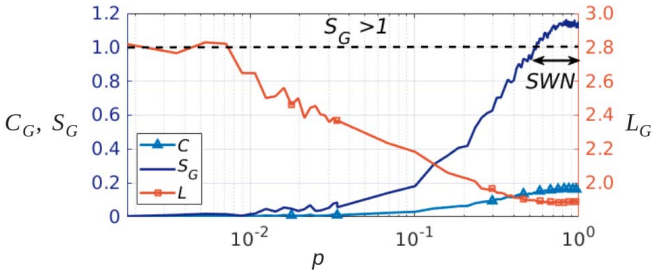

Fig. 5. Clustering coefficient (C), small-world property (S_G), and path length (L) versus rewiring probability. The region where the graph transforms into a small-world network is shown with the double-headed arrow.

b) Network profiling: Using the aforementioned rewiring policy, we generate various graphs by sweeping the rewiring probability in the [0,1] interval. Fig. 5 demonstrates the correlation between C and L as the rewiring probability is changed for a 14-layer DNN. For conventional DNNs, the clustering coefficient is zero and the characteristic path length can be quite large specifically for very deep networks (leftmost points on Fig. 5). As such, DNNs are far from networks with the small-world property. Random rewiring replaces short-range connections between subsequent layers with longer-range connections. Consequently, L is reduced while C increases as the network shifts towards its small-world equivalent. We select the topology with the maximum value of small-world property, S_G, as the SWANN. As a direct result of such architectural modification, the new network enjoys enhanced connectivity in its dataflow graph which results in better gradient propagation and training speedup.

To compare the training convergence of SWANN with other configurations generated during the probability sweep, we train several rewired networks on the MNIST dataset [26], each of which is constructed from a 5-layer DNN. Fig. 6 demonstrates the convergence rate of these various architectures versus the rewiring probability p that is used to generate them from the baseline DNN. Due to the addition of long-range connections, all models show convergence improvements over the baseline. However, the perfect balance between node clustering and average path length is achieved for the SWN which leads to the fastest training convergence for SWANN.

C. SWANN Methodology

1) DNN Formulation: Conventional DNNs are comprised of subsequent layers where each layer, l, in the network performs

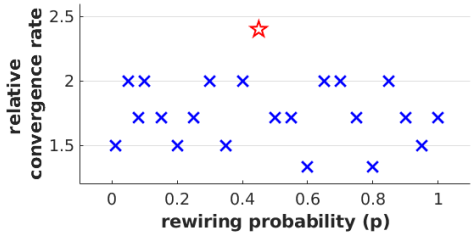

Fig. 6. Convergence to 99.0% test accuracy for a 5-layer DNN and its randomly rewired counterparts trained on the MNIST dataset. Here, the relative convergence rate is computed as $\frac{e_b}{e_r}$, where e_b and e_r denote the number of training epochs required for the baseline and rewired models to reach the target accuracy, respectively. The SWN is shown with a red star.

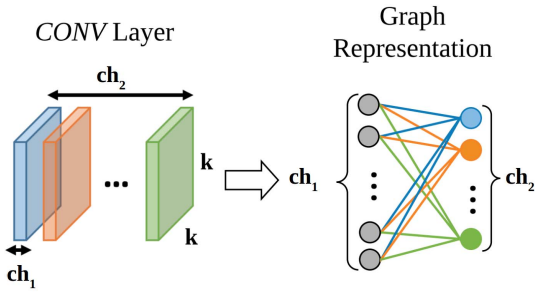

Fig. 7. Conversion of a *CONV* layer to its graph representation. Each $k \times k$ convolution kernel is replaced by an edge in the corresponding graph where the input and output filter channels are shown as two consecutive rows of vertices with ch_1 and ch_2 nodes, respectively.

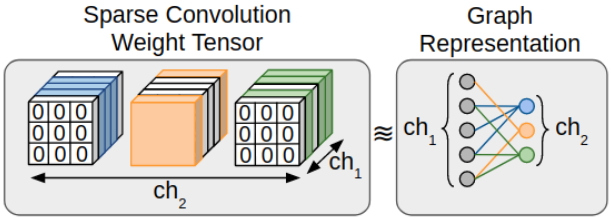

Fig. 8. Coarse-grained sparse convolution between a layer with $ch_1 = 5$ output channels and a layer with $ch_2 = 3$ output channels. Left: Sparse convolution weights. For each removed connection from the graph, we show the corresponding filter in the sparse convolution weight by zero. The colored channels represent trainable DNN weights which can take on any arbitrary floating-point value. Right: Equivalent graph with nodes representing channels.

a combination of linear and nonlinear operations on its input, x_l, to generate the corresponding output, y_l. We denote core linear operations ($CONV$ and FC) in a DNN by $W_l(\cdot)$ with the subscript representing the layer index. Other operations can take the form of Batch Normalization (BN), $ReLU$ activation, and Pooling. For each linear layer, we bundle one or more of such operations together and show them as one composite function, $C_l(.)$. For an arbitrary layer l in a conventional DNN, the output is thus formalized as:

$$y_l = C_l(W_l(x_l)) \tag{3}$$

Note that the cascaded nature of DNNs implies that the generated output from one layer serves as the input to the immediately succeeding layer, i.e., $x_{l+1} = y_l$.

2) Sparse Connections in SWANNs: One major difference between SWANNs and conventional DNNs is that SWANN layers can be interconnected regardless of their position in the network hierarchy. More specifically, the output of each layer of a SWANN is connected to all its succeeding layers via sparse weight tensors. These connections are implemented via convolution kernels with coarse-grained sparsity patterns.

By definition, *CONV* layers sweep a $k \times k$ convolution kernel across an input tensor of dimensionality $W_{in} \times H_{in} \times ch_1$ to generate an output feature map with dimensions $W_{out} \times H_{out} \times ch_2$ where ch_1 and ch_2 denote the number of input and output channels, respectively. In order to generate the graph-equivalent of such layer, we represent each $k \times k$ kernel by a single edge in the graph as shown in Fig. 7. To remove each connection from the graph, we mask the corresponding $k \times k$ kernel to zero to generate a sparse weight tensor. Fig. 8 shows the convolution filters of an example sparse connection from a layer with 5 output channels to a layer with 3 output channels and the corresponding small-world graph representation.

Let us denote sparse connections from layer l_1 to layer l_2 by $W^s_{l_1 l_2}(.)$. The output of the l-th layer in SWANN can then be calculated as:

$$y_l = C_l(W^s_l(x_l) + \sum_{l_1 < l-1} W^s_{l_1\ l}(y_{l_1})) \tag{4}$$

Comparing the above formulation with Eq. 3, we highlight the extra summation term that accounts for the inter-layer connections. Note that in Eq. 4, both W^s_l and $W^s_{l_1\ l}$ are sparse tensors. The inter-layer connectivity in SWANN enables enhanced data flow, both during inference and training stages, while the sparse connections mitigate unnecessary parameter utilization. Unlike the previously proposed feature concatenation methodology [1], we perform summation over the feature-maps. This, in turn, mitigates the appearance of extremely high dimensional weight kernels that result from channel-wise feature-map concatenation. Furthermore, the summation of feature-maps enables SWANN to be applicable to all network architectures with various layer configurations. We gradually increase the stride in the long-range sparse connections as a function of the distance between the inter-linked layers. This allows us to reduce the dimensionality of the produced feature-maps as well as tune the impact of added long-range connections. In addition to adjusting the convolution strides, we use zero-padding where necessary to match the dimensionality of inter-layer connected feature-maps.

3) Composite Non-Linear Operation: Unlike DenseNets [1] and ResNets [9] where several linear layers are concatenated before pooling is performed, SWANNs support pooling immediately after each $CONV$ layer as seen in conventional DNN architectures. We experiment with various configurations of the widely-used non-linear operations, i.e., BN, $ReLU$, and $Maxpool$ to investigate the effect of ordering on network convergence. Our experiments demonstrate that SWANN convergence is enhanced when the composite non-linear function, C_l is implemented as a $ReLU$, followed by $Maxpool$ing and BN as shown in Fig. 2.

TABLE I

BENCHMARKED DNNS FOR EVALUATING SWANN EFFECTIVENESS. $CONV$ LAYERS ARE REPRESENTED AS $\langle kernel\ size \rangle$CONV AND FC LAYERS ARE DENOTED BY $\langle output\ elements \rangle$FC. BN AND $ReLU$ ARE NOT SHOWN FOR BREVITY

	Convolution	MaxPool	Convolution	MaxPool	Convolution	MaxPool	Convolution	MaxPool	Convolution	MaxPool	Classifier
MNIST	5×5 Conv	2×2 stride 2	5×5 Conv	2×2 stride 2	5×5 Conv	2×2 stride 2	-	-	-	-	84FC 10FC, softmax
ConvNet-C [27]* (C10, C100)	[3×3 Conv] $\times 2$	2×2 stride 2	[3×3 Conv] $\times 2$	2×2 stride 2	[3×3 Conv] $\times 3$	2×2 stride 2	[3×3 Conv] $\times 3$	2×2 stride 2	[3×3 Conv] $\times 3$	2×2 stride 2	512FC 10FC, softmax
AlexNet [28] (ImageNet)	11×11 Conv (stride 4)	2×2 stride 2	5×5 Conv	2×2 stride 2	3×3 Conv	-	3×3 Conv	-	3×3 Conv	2×2 stride 2	4096FC 4096FC 1000FC, softmax
ResNet-18 [9] (ImageNet)	7×7 Conv (stride 2)	3×3 stride 2	[3×3 Conv] $\times 4$	-	[3×3 Conv] $\times 4$	-	[3×3 Conv] $\times 4$	-	[3×3 Conv] $\times 4$	7×7 average pool	1000FC, softmax

* We modify the ConvNet-C fully-connected layers form [27] to comply with the CIFAR datasets.

	Convolution	Dense Block (1)	Transition Block	Dense Block (2)	Transition Block	Dense Block (3)	Classifier
DenseNet-40 [1] (**C10**)	3×3 Conv	[3×3 Conv]$\times 12$	1×1 Conv 2×2 average pool	[3×3 Conv]$\times 12$	1×1 Conv 2×2 average pool	[3×3 Conv]$\times 12$	8×8 average pool 10FC, softmax

* Conv denotes a BN, followed by a $ReLU$ and a convolution layer.

IV. EXPERIMENTS

We conduct proof-of-concept experiments on different network architectures and image classification benchmarks to empirically demonstrate the enhanced training convergence of SWANNs compared to the baseline (conventional) DNNs. We leverage popular DL libraries Keras and PyTorch for our implementations. All experiments are performed on a machine with Nvidia Titan XP GPU and Intel Xeon CPU.

Our evaluations target both centralized and decentralized on-device learning scenarios. Sec. IV-C, IV-D, IV-E enclose our evaluations in the centralized training setup. This setup directly simulates on-device learning applications where users train/finetune a model locally on their personal data samples, e.g., personalization. Sec. IV-F encloses the evaluation in the decentralized (federated) learning setup where several users collaboratively train a global model by performing multiple local updates and a global aggregation.

A. Datasets

1) MNIST: This dataset consists of 10 classes of 28×28 gray-scale images from handwritten digits with 60,000 train and 10,000 test images. We normalize the data using the per-channel mean and variance prior to training and testing.

2) CIFAR: We carry out our experiments on the two available CIFAR datasets. CIFAR10 (C10) and CIFAR100 (C100) benchmarks consist of colored images with dimensionality 32×32 that are categorized in 10 and 100 classes, respectively. Each dataset contains 50,000 samples for training and 10,000 samples for testing. We use standard data augmentation routines popular in prior work [9], [10]: samples are normalized using per-channel mean and standard deviation. At training time, random horizontal mirroring, shifting, and slight rotation are also applied.

3) ImageNet: The ISLVRC-2012 dataset, widely known as the ImageNet, consists of 1000 different classes of colored images with 1.2 million samples for training and 50,000 samples for validation. We use the augmentation scheme proposed in [27] and [29] to preprocess input samples: during training, we resize the images by randomly sampling the shorter edge from [256, 480]. A 224×224 crop is then randomly sampled from the image. We also perform per-channel normalization as well as horizontal mirroring [28].

TABLE II

GRAPH CHARACTERISTICS OF SWANN MODELS

model	γ_G	λ_G	S_G	p
MNIST	1.09	0.99	1.09	0.45
ConvNet-C	1.20	1.01	1.19	0.85
AlexNet	1.48	1.00	1.48	0.80
ResNet-18	1.06	1.00	1.06	0.83
DenseNet-40	1.20	1.01	1.19	0.92

B. Benchmarked Architectures

Tab. I encloses our baseline DNN architectures. SWANNs maintain the same feed-forward architecture as the baseline networks and are constructed by 1) replacing $CONV$ layers with sparse convolutions and 2) adding sparse convolutions between non-consecutive layers. Table II encloses the relative clustering coefficient γ_G, relative average path length λ_G, small-world property S_G, and rewiring probability p to achieve the corresponding SWANN for each baseline DNN. Here, γ_G, λ_G, and S_G follow the definitions in Eq. (2). As seen, all SWANN models satisfy the small-world characteristic, i.e., $S_G > 1$.

C. Results on MNSIT

We train the 5-layer architecture shown in Tab. I as our baseline. The small-world equivalent of the baseline model is generated using a rewiring probability of 0.5. To prevent overfitting, a dropout layer with the rate of 0.5 is added between the two FC layers in both the baseline DNN and SWANN. We train the models using Adadelta optimizer [30] with an initial learning rate of 1 and a decay of 0.95. Batch size is set to 256 for both models.

1) Convergence: Figure 9 compares the test error and training loss of the baseline DNN and its small-world counterpart throughout training. Both models achieve a final test accuracy of 99.1% on the MNIST dataset. The plain baseline DNN converges to the aforesaid accuracy in 19 epochs (=4864 iterations) while SWANN reaches the convergence accuracy in 9 epochs (=2304 iterations) which shows 2.1× improvement over the baseline.

D. Results on CIFAR

1) ConvNet-C: We train the ConvNet-C [27] model on C10 and C100 benchmarks with a batch size of 128. To prevent

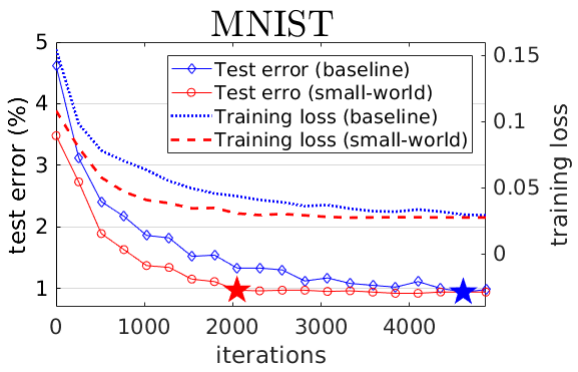

Fig. 9. Comparison of a plain DNN's training convergence with its small-world equivalent. Here, the red and blue colors show SWANN and baseline, respectively. The $\star$ markers denote the point of convergence to final test accuracy for the models with the corresponding colors.

TABLE III

POINT-WISE COMPARISON OF CONVERGENCE SPEED-UP FOR A SWANN AND ITS EQUIVALENT BASELINE NETWORK (CONVNET-C) ON CIFAR BENCHMARKS

CIFAR10	**Baseline**	Test Error (%)	24.21	17.80	9.22	8.51	**7.56**
		Iterations	1408	2560	8704	11008	18560
	SWANN	Test Error (%)	23.73	17.57	8.64	8.25	**7.44**
		Iterations	896	1536	4992	5888	7040
	Speed-up		1.57×	1.67×	1.74×	1.87×	**2.64×**
CIFAR100	**Baseline**	Test Error (%)	77.08	52.3	41.54	31.14	**29.52**
		Iterations	2944	6144	9472	16128	28928
	SWANN	Test Error (%)	76.67	50.57	40.18	31.15	**29.26**
		Iterations	384	1408	3072	7808	10240
	Speed-up		7.67×	4.36×	3.08×	2.1×	**2.82×**

overfitting, a dropout layer with a rate of 0.5 is added before the first FC layer. The small-world model is constructed using the same configuration of layers as the baseline, including the dropout layers. We use Stochastic-Gradient-Descent (SGD) optimizer with Nesterov, 0.9 momentum, and a $5e-4$ weight decay. Models are trained for $2e{+}4$ and $3e{+}4$ iterations on C10 and C100, respectively. The initial learning rate is set to 0.01 for both datasets and learning rate is decayed by 0.5 upon optimization plateau.

a) Convergence: Fig. 10-(a) illustrates the test error and training loss of the baseline and SWANNs as two representatives of the convergence speed. Similarly, for C100 benchmark, the corresponding convergence curve is presented in Fig. 10-(b). While these figures qualitatively demonstrate the effectiveness of our methodology, we provide a quantitative measure for a solid comparison between SWANN and the baseline. We investigate several points corresponding to various test accuracies and compare the two models' convergence to these points.

Tab. III summarizes the per-accuracy speed-up of SWANN over the baseline model. As seen, the speed-up varies for different accuracies, however, for all test accuracies, SWANN requires a substantially fewer number of iterations for convergence. At the final saturation point (marked by $\star$ on Fig. 10), both models achieve comparable accuracies while SWANN enjoys a 2.6× and 2.8× reduction in convergence time for C10 and C100 datasets, respectively.

TABLE IV

COMPARISON OF THE COMPUTATIONAL COMPLEXITY AND MODEL PARAMETER SPACE BETWEEN A 40-LAYER DENSENET WITH $k = 12$ AND THE CORRESPONDING SWANN

Model	**Depth**	**Params**	**FLOPs**	**Test Error**
DenseNet ($k = 12$)	40	910K	285.3M	0.071
SWANN	40	98K	85.5M	0.074

2) DenseNet: DenseNets [1] achieve state-of-the-art accuracy by connecting all neurons from different layers of a dense block with trainable (dense) parameters. Such dense connectivity pattern results in high redundancy in the parameter space and causes extra overhead on training. We show that a SWANN with only sparse connections and much fewer parameters achieves similar results as DenseNet.

We train a DenseNet model with 40 layers and $k = 12$ (Tab. I) on C10 dataset. The equivalent SWANN is constructed by removing all long-range dense connection from the architecture and rewiring the remaining (short) edges such that each dense block becomes small-world. The SWANN maintains the same number of layers while the inter-layer connections are implemented using sparse convolution kernels, thus incurring substantially fewer number of trainable parameters.

We use the publicly available PyTorch implementation for DenseNets[1] and replace the model with our small-world network. Same training scheme as explained in the original DenseNet paper [1] is used: models are trained for 19200 iterations with a batch size of 64. Initial learning rate is 0.1 and decays by 10 at $\frac{1}{2}$ and $\frac{3}{4}$ of the total training iterations.

a) Convergence: Fig. 11 demonstrates the test accuracy of the models versus the number of epochs. As can be seen, although SWANN has much fewer parameters, both models achieve comparable validation accuracy while showing identical convergence speed. We report the computational complexity (FLOPs) of the models as the total number of multiplications performed during a forward propagation through the network. Tab. IV compares the benchmarked DenseNet and SWANN in terms of FLOPs and number of trainable weights in $CONV$ and FC layers. We highlight that SWANN achieves comparable test accuracy while having 10× reduction in parameter space size.

E. Results on ImageNet

1) AlexNet: We train the AlexNet [28] model on ImageNet dataset and follow the architecture provided in the Caffe model zoo [31] (See Tab. I). In order to mitigate overfitting, we add dropout layers with probability 0.5 after the FC layers. Loss minimization is performed by means of SGD with Nesterov [32] and a 0.9 momentum. We set the batch size to 64 for both models and incorporate an exponential decay for the learning rate: initial learning rate is set to $2.5e{-}3$ and the decay factor is 0.99999875 [33].

a) Convergence: To fully examine the performance of our model, we report the speed-up of SWANN over the baseline for several values of test error. Tab. V encloses the point-wise comparison between the benchmarked models. As can be

[1] https://github.com/andreasveit/densenet-pytorch

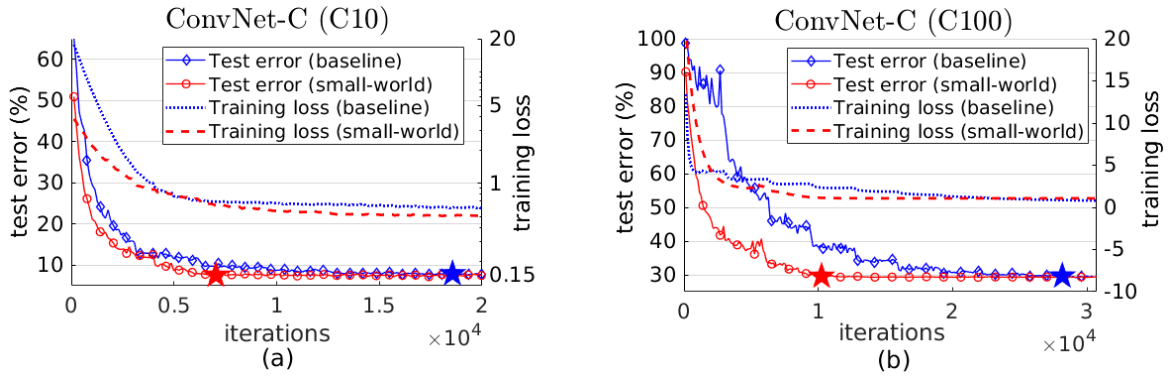

Fig. 10. Test error and training loss versus iterations for a ConvNet-C model and the rewired SWANN trained on (a) CIFAR10 and (b) CIFAR100 datasets. Here, the red and blue colors show SWANN and baseline, respectively. The $\star$ markers denote the point of convergence to final test accuracy for the models with the corresponding colors.

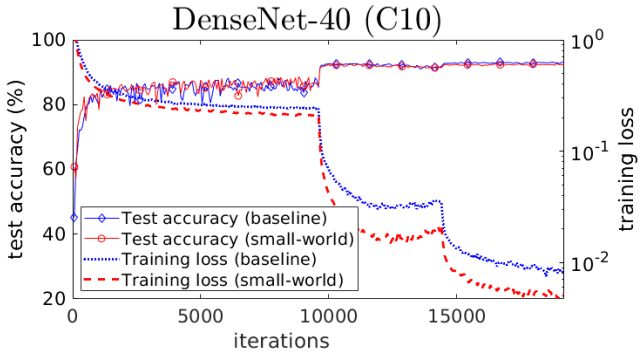

Fig. 11. Training loss and testing accuracy of the 40-layer ($k = 12$) DenseNet [1] with 1M parameters and our corresponding SWANN with less than 100K parameters.

TABLE V

PERFORMANCE OF A BASELINE ALEXNET AND ITS SWANN BENCHMARKED ON IMAGENET DATASET

AlexNet	**Baseline**	Test Error (%)	51.72	46.29	44.21	42.31	**42.01**
		Iterations	1088	2304	3264	4416	5120
	SWANN	Test Error (%)	51.97	46.49	44.25	42.31	**41.55**
		Iterations	768	1664	2368	3520	3776
	Speed-up		1.42×	1.38×	1.38×	1.25×	**1.36×**

seen, for all values of test error, the convergence of our small-world architecture is faster. SWANN converges to the final test accuracy after 3776 iterations while the baseline model needs 5120 iterations, resulting in a 1.4× overall speed-up.

2) ResNet: We adopt the training scheme in the original ResNet paper [9]. To build the SWANN, we first remove all shortcut and bottleneck connections from the model. We then rewire the connections in the acquired plain network such that it becomes small-world. No dropout is used for the baseline and SWANN. Batch size is set to 128 and we use SGD with 0.9 momentum and $1e-4$ weight decay. Initial learning rate is 0.1 and decays by 0.1 when the accuracy plateaus. We train the models for $9e+5$ iterations and report single-crop accuracies.

a) Convergence: We enclose point-wise comparisons between the baseline ResNet and SWANN for various iterations and test errors in Tab. VI. As seen, SWANN achieves faster convergence throughout training and reaches the final test accuracy with 1.3× less iterations. This shows that the systematic restructuring of long edges in SWANN allows for a better convergence behavior compared to the replicated blocks in ResNet.

TABLE VI

POINT-WISE CONVERGENCE COMPARISON OF A BASELINE RESNET-18 AND ITS SWANN EQUIVALENT ON IMAGENET DATASET

ResNet-18	**BaseLine**	Test Error (%)	60.37	56.94	37.91	**31.72**
		Iterations	1792	3456	7424	9344
	SWANN	Test Error (%)	59.63	56.76	37.86	**31.68**
		Iterations	512	768	3584	7168
	Speed-up		3.50×	4.50×	2.07×	**1.31×**

F. Federated Learning

In this section, we corroborate SWANN enhanced convergence in the federated learning setting as a candidate application for on-device learning. In this setup, a server holds a global model and users each have a unique (local) dataset. The users compute the weight updates for the global model on their local datasets and communicate the updated weights to the server. The server then aggregates the weights from all users and updates the global model. We implement the popular `FedAvg` algorithm [34] for federated learning where each user performs several local updates on the global model before sending the updated weights to the global server.

Following the original paper [34] we consider a pool of 100 users and randomly select 10 users at each iteration to send their updates to the server. We perform 5 local updates on each user with a batch size of 10. This setting provides a good balance between the total number of communication rounds for convergence and the required amount of computation per iteration for each user [34]. Optimization is performed using SGD with learning rate of 0.01 for both the baseline DNN and the SWANN. Our evaluations are performed on the MNIST dataset with the baseline DNN architecture shown in Tab. I.

We evaluate SWANN under two distributed data settings: 1) IID where the data is distributed uniformly across all users, i.e., each user has instances of all classes. 2) Non-IID where the entire data is sorted by their label, divided into 200 shards

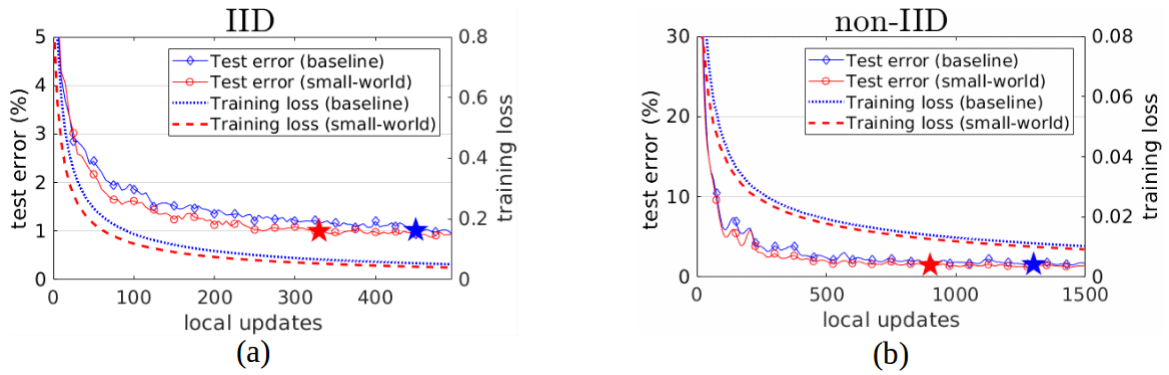

Fig. 12. Training loss and testing accuracy of the baseline 5-layer DNN and SWANN in the federated learning scenario with a) IID and b) non-IID data distributions. Here, the red and blue colors show SWANN and baseline, respectively. The ⋆ markers denote the point of convergence to final test accuracy for the models with the corresponding colors.

of size 300, and each user is assigned 2 shards [34]; users on average only have instances from two classes. Fig. 12-a,b demonstrate the test accuracy and training loss versus number of local updates for the baseline DNN and its corresponding SWANN in the IID and non-IID settings, respectively.

1) IID Data: To reach the final test accuracy of 99.1%, the baseline DNN requires 450 local updates while SWANN only requires 330, thereby showing a 1.4× reduction in the computation performed on the user devices. In addition to the savings in computation, SWANN also reduces the total number of required global aggregations by 1.4×, which directly translates to a reduced communication cost between users and the server.

2) Non-IID Data: In the non-IID setting, SWANN outperforms the baseline DNN both in terms of the convergence rate and final test accuracy. The baseline DNN requires 1300 local updates to converge to the final test accuracy of 98.7% while SWANN converges to 98.9% test accuracy with 1020 local updates. As a result, SWANN achieves 1.3× reduction in the users' computation and communication during training. The higher accuracy of SWANN in the challenging non-IID data setting further demonstrates the better stability of SWANN during training compared to non small-world architectures.

For a more detailed comparison, we include additional convergence points to target test accuracies for the baseline DNN and SWANN in Tab. VII. As shown, SWANN reaches all target accuracies considerably faster than the baseline. On average, SWANN requires 1.5× and 1.6× less computation and communication for embedded user devices in the IID and non-IID settings, respectively.

V. Discussion on Long-Range Connections

The selected small-world structure for a given DNN has two main characteristics, namely high clustering of nodes and small average path length between neurons across layers. We postulate that such qualities render the SWN desirable during training due to the enhanced information flow paths existent in these efficiently-connected networks. To examine our hypothesis, we visualize the weights connecting different layers of the trained SWANN for C10, C100 (ConvNet-C), and

TABLE VII

Point-Wise Convergence Comparison of the 5-Layer Baseline DNN and Its Corresponding SWANN in the Federated Learning Scenario

IID	BaseLine	Test Error (%)	98.16	98.65	98.90	**99.07**
		Local Updates	85	200	355	450
	SWANN	Test Error (%)	98.14	98.65	98.89	**99.07**
		Local Updates	60	130	220	330
	Speed-up		1.42×	1.54×	1.61×	**1.36×**
non-IID	BaseLine	Test Error (%)	96.12	96.67	98.10	**98.74**
		Local Updates	170	405	555	1300
	SWANN	Test Error (%)	96.14	97.63	98.16	**98.72***
		Local Updates	100	230	325	900
	Speed-up		1.70×	1.76×	1.71×	**1.44×**

*SWANN reaches a final test accuracy of 98.90% after 1020 local updates.

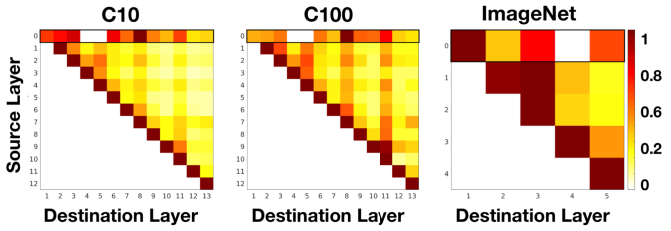

Fig. 13. Visualization of average absolute value of trained weights within $CONV$ layers of SWANNs. Colors encode the connectivity strength between layers with red being the strongest and white denoting no connection. The marked rows with black box borders correspond to the input layer of the networks.

ImageNet (AlexNet) benchmarks. Fig. 13 presents a heat map of the average absolute values of weights connecting each pair of $CONV$ layers.

Each square at position (l_1, l_2) of the heatmap represents the strength of the connections between layers l_1 and l_2 where l_0 denotes network input. Color shades of orange, red and maroon indicate strong inter-layer dependency while the white color indicates that no connections are present between the corresponding layers in SWANN. We summarize our observations based on the heat map as the following:

1) Each layer has strong connections to its non-subsequent layers indicating that long-range edges established in SWANN are crucial to performance.

2) The input layer has spread weights across all layers of the network which demonstrates the importance of connections between earlier and deeper layers.
3) SWANN preserves the strong connections between one layer and the immediately proceeding layer, thus, maintaining the conventional DNN data flow.

VI. Conclusion and Future Work

We propose a novel methodology that adaptively modifies conventional feed-forward DL models to new architectures, called SWANN, that fall into the category of small-world networks—a class of complex graphs used to study real-world models such as human brain and the neural networks of animals. By leveraging the intriguing features of small-world networks, e.g., enhanced signal propagation speed and synchronizability, SWANNs enjoy improved data flow within the network, resulting in substantially faster convergence speed during training. Our small-world models are implemented via sparse connections from each DNN layer to all succeeding layers. Such sparse convolutions enable SWANNs to benefit from long-range connections while mitigating the redundancy in the parameter space existent in prior art.

As our experiments demonstrate, SWANNs are able to achieve state-of-the-art accuracy in $\approx 2.1\times$ lower number of training iterations, on average. Furthermore, compared to a densely-connected architecture, SWANNs achieve comparable accuracy while having $10\times$ reduction in the number of parameters. In summary, due to their optimal graph connectivity and fast response to training, SWANNs can be advantageous for on-device learning in embedded applications.

Current implementation of SWANN leverages regular dense operations from the PyTorch library to realize sparse convolutions. As such, we see an average of $\sim 27.7\%$ increase in the inference runtime due to the overhead of added connections. However, ongoing advances in both software and hardware fronts provide several new opportunities to take advantage of the high sparsity in SWANNs and achieve performance improvements. Developing specialized deep learning compilers and leveraging the state-of-the-art sparse compute kernels [35] are exciting directions to be explored in future work.

References

[1] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, p. 3.

[2] S. Xie, A. Kirillov, R. Girshick, and K. He, "Exploring randomly wired neural networks for image recognition," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1284–1293.

[3] M. Wortsman, A. Farhadi, and M. Rastegari, "Discovering neural wirings," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 2684–2694.

[4] J. You, J. Leskovec, K. He, and S. Xie, "Graph structure of neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 10881–10891.

[5] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, p. 440, 1998.

[6] R. Olfati-Saber, "Ultrafast consensus in small-world networks," in *Proc. Amer. Control Conf.*, Jun. 2005, pp. 2371–2378.

[7] A. Tahbaz-Salehi and A. Jadbabaie, "Small world phenomenon, rapidly mixing Markov chains, and average consensus algorithms," in *Proc. 46th IEEE Conf. Decis. Control*, Dec. 2007, pp. 276–281.

[8] D. H. Zanette, "Dynamics of rumor propagation on small-world networks," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 65, no. 4, Mar. 2002, Art. no. 041908.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.

[10] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, Oct. 2016, pp. 646–661.

[11] G. Huang, S. Liu, L. van der Maaten, and K. Q. Weinberger, "CondenseNet: An efficient densenet using learned group convolutions," *Group*, vol. 3, no. 12, p. 11, 2017.

[12] G. Krishnan, Y. Ma, and Y. Cao, "Small-world-based structural pruning for efficient FPGA inference of deep neural networks," in *Proc. IEEE 15th Int. Conf. Solid-State Integr. Circuit Technol. (ICSICT)*, Nov. 2020, pp. 1–5.

[13] M. Tan and Q. V. Le, "EfficientNetV2: Smaller models and faster training," 2021, *arXiv:2104.00298*.

[14] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 4780–4789.

[15] K. Bhardwaj, G. Li, and R. Marculescu, "How does topology of neural architectures impact gradient propagation and model performance?" in *Proc. CVPR*, 2021, pp. 13498–13507.

[16] B. H. Ahn, J. Lee, J. M. Lin, H.-P. Cheng, J. Hou, and H. Esmaeilzadeh, "Ordering chaos: Memory-aware scheduling of irregularly wired neural networks for edge devices," in *Proceedings of Machine Learning and Systems*, vol. 2, I. Dhillon, D. Papailiopoulos, and V. Sze, Eds. 2020, pp. 44–57. [Online]. Available: https://proceedings.mlsys.org/paper/2020/file/9bf31c7ff062936a96d3c8bd1f8f2ff3-Paper.pdf

[17] D. Simard, L. Nadeau, and H. Kröger, "Fastest learning in small-world neural networks," *Phys. Lett. A*, vol. 336, no. 1, pp. 8–15, Feb. 2005.

[18] O. Erkaymaz, M. Ozer, and M. Perc, "Performance of small-world feedforward neural networks for the diagnosis of diabetes," *Appl. Math. Comput.*, vol. 311, no. 15, pp. 22–28, Oct. 2017.

[19] L. Xiaohu, L. Xiaoling, Z. Jinhua, Z. Yulin, and L. Maolin, "A new multilayer feedforward small-world neural network with its performances on function approximation," in *Proc. IEEE Int. Conf. Comput. Sci. Automat. Eng.*, Jun. 2011, pp. 353–357.

[20] S. H. Strogatz, "Exploring complex networks," *Nature*, vol. 410, no. 6825, p. 268, 2001.

[21] V. Latora and M. Marchiori, "Efficient behavior of small-world networks," *Phys. Rev. Lett.*, vol. 87, Oct. 2001, Art. no. 198701.

[22] M. Barahona and L. M. Pecora, "Synchronization in small-world systems," *Phys. Rev. Lett.*, vol. 89, no. 5, Jul. 2002, Art. no. 054101.

[23] M. Kuperman and G. Abramson, "Small world effect in an epidemiological model," *Phys. Rev. Lett.*, vol. 86, no. 13, p. 2909, 2001.

[24] X. Zhang, C. Moore, and M. E. J. Newman, "Random graph models for dynamic networks," *Eur. Phys. J. B*, vol. 90, no. 10, p. 200, Oct. 2017.

[25] M. D. Humphries and K. Gurney, "Network 'small-world-ness': A quantitative method for determining canonical network equivalence," *PLoS ONE*, vol. 3, no. 4, Apr. 2008, Art. no. e0002051.

[26] Y. LeCun. (1998). *The MNIST Database of Handwritten Digits*. [Online]. Available: https://yann.lecun.com/exdb/mnist/

[27] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 630–645.

[30] M. D. Zeiler, "ADADELTA: An adaptive learning rate method," 2012, *arXiv:1212.5701*.

[31] *BAIR/BVLC AlexNet Model*. Accessed: Sep. 31, 2018. [Online]. Available: https://github.com/BVLC/caffe/tree/master/models/bvlc_alexnet

[32] Y. Nesterov *et al.*, "Gradient methods for minimizing composite objective function," *Math. Program.*, vol. 140, no. 1, pp. 125–161, 2013.

[33] L. N. Smith, "Cyclical learning rates for training neural networks," in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Mar. 2017, pp. 464–472.

[34] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*. PMLR, 2017, pp. 1273–1282.

[35] J. Choquette, W. Gandhi, O. Giroux, N. Stam, and R. Krashinsky, "NVIDIA A100 tensor core GPU: Performance and innovation," *IEEE Micro*, vol. 41, no. 2, pp. 29–35, Mar. 2021.

Mojan Javaheripi (Graduate Student Member, IEEE) received the B.Sc. degree in electrical engineering from the Sharif University of Technology in 2017 and the M.Sc. degree in computer engineering from UC San Diego in 2020, where she is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering. Her research interests include deep learning performance optimization, safe and robust deep learning, and SW/HW co-design for machine learning.

Bita Darvish Rouhani (Student Member, IEEE) received the B.Sc. degree in electrical engineering from the Sharif University of Technology in 2013, the M.Sc. degree in electrical and computer engineering from Rice University in 2015, and the Ph.D. degree in computer engineering from UC San Diego. Her research interests include deep learning, safety of machine learning models, low-power computing, distributed optimization, and big data analysis.

Farinaz Koushanfar (Fellow, IEEE) received the B.Sc. degree in electrical engineering from the Sharif University of Technology, the M.S. degree in electrical engineering from UCLA, and the Ph.D. degree in electrical engineering and computer science and the M.A. degree in statistics from UC Berkeley in 2005. She is currently a Professor and a Henry Booker Faculty Scholar of electrical and computer engineering with University of California San Diego, where she is directing the Adaptive Computing and Embedded Systems (ACES) Lab. Before joining the faculty at UC San Diego, she was a Professor of electrical and computer engineering with Rice University. Her research interests include embedded and cyber-physical systems design, embedded systems security, and design automation of domain-specific/mobile computing and machine learning applications. She is a fellow of the Kavli Foundation Frontiers of the National Academy of Engineering. She has received a number of awards and honors for her research, mentorship, and teaching, including the Presidential Early Career Award for Scientists and Engineers (PECASE) from President Obama, the ACM SIGDA Outstanding New Faculty Award, the MIT TR-35, and the Young Faculty/CAREER Award from NSF, DARPA, ONR, and ARO.