An Overview of Adaptive Dynamic Deep Neural Networks via Slimmable and Gated Architectures

1st Timothy K Johnsen
University of California Irvine, USA
tjohnsen@uci.edu

2nd Ian Harshbarger
University of California Irvine, USA
iharshba@uci.edu

3rd Marco Levorato
University of California Irvine, USA
levorato@uci.edu

***Abstract*—Deep Neural Networks (DNN) are omnipresent in systems that are developed for processing vast quantities of data, most particularly images, for tasks such as perception, navigation, classification, detection, and segmentation. Such DNNs can be computationally demanding from the high number, upwards to hundreds of billions, of computations required during execution. Dynamic Deep Neural Networks (DDNN) are an evolution that allow the number of computations in a DNN to be scaled down. However, lacking in DDNN methodologies is the functionality to control online when and how to scale down the number of computations. To respond to this need, Adaptive Dynamic Deep Neural Networks (ADDNN) are an evolving class of deep learning models used in high performance computing that attempt to minimize resources usage – memory and power – and latency while maintaining an acceptable task performance by adapting the model architecture in response to the current context. Thus, ADDNNs are a State-of-the-Art evolution that perceive context, on a case-by-case basis, and adapt the number of computations to that required by the difficulty of the problem at hand. This positional paper is not only a survey of current DDNN methods in literature, but also an analysis into the considerations, design principles and challenges in developing robust ADDNN systems and applications.**

***Index Terms*—Dynamic Deep Neural Networks; Computer Vision, Multi-Branch Neural Networks, Slimmable Deep Neural Networks; Gated Deep Neural Networks.**

I. Introduction

Deep Neural Networks (DNN) are a class of data-driven algorithmic models achieving high performance at processing high-dimensional data structures. The most prominent application of DNNs is computer vision, where the model is performing tasks such as classification [1], [2], object detection [3], and segmentation [4]. Such computer vision models can be further used as perception modules in downstream tasks, for example as in robotics and autonomy [5]. The functionality of DNN is based on revealing a large latent feature space, extracted from the presented data, that encapsulates generalized correlations which are then leveraged to process novel data. This feature space is realized in the form of upwards to hundreds of billions of trainable model parameters, resulting in a similar amount of computations required to execute the DNN. Consequently, large State-of-the-Art (SoA) DNN models can perform with exceptionally low error rates, even outperforming humans [6]. However, the maximum number of feasible computations is limited by the computing resources available to the computers that are being used to host the DNN.

The standard approach to address the limited computing resources available in many application settings is model reduction: a static approach to reduce the computational overhead required to execute a DNN by directly adjusting its structure before runtime. The main strategies used to reduce the computational needs of DNN models are quantization, direct design and model compaction. Quantization [7]–[9] transforms high precision floating point numbers (high number of bits) into low precision ones (low number of bits), reducing the execution time and memory required to execute a DNN. Direct design [10] trains a DNN from scratch while employing methods that force a lower number of parameters from the beginning. Model compaction is used to reduce the number of parameters of an already trained model, such as pruning [11] and knowledge distillation [12]–[14]. The resulting models typically incur a degradation in task performance. Implementation optimization directly improves how the neural network is computed on the processing unit, such as in [15].

In Edge computing [16], [17] in order to reduce the computational load at mobile devices, the execution of the models is offloaded to a wirelessly connected compute-capable device. In this computing paradigm, the mobile device acquires and compresses the data. The data are then sent to the edge server, decompressed and fed into the DNN. The critical aspect of edge computing is the transmission of potentially large volumes of data over volatile and capacity-constrained communication channels, which leads to issues such as increased latency, data loss, and security.

Recent methods focus on creating a dynamic computing framework to reducing the computational load required to execute a DNN, allowing its structure to be changed after its design and training, creating what we refer to as a Dynamic DNN (DDNN). A natural evolution to a DDNN, is to include the functionality to intelligently decide how and when to change the structure as to scale down its complexity to only that required by the situation at hand (that is, the input data), in what we refer to as an Adaptive DDDN (ADDNN). The remainder of this paper presents an overview of design principles, challenges, and applications of both DDNNs and ADDNNs.

Specifically, a DDNN can be classified as either width scaling, where the number of computations in each layer is reduced, or depth scaling, where the number of layers are reduced. The core challenge in developing a DDNN is how

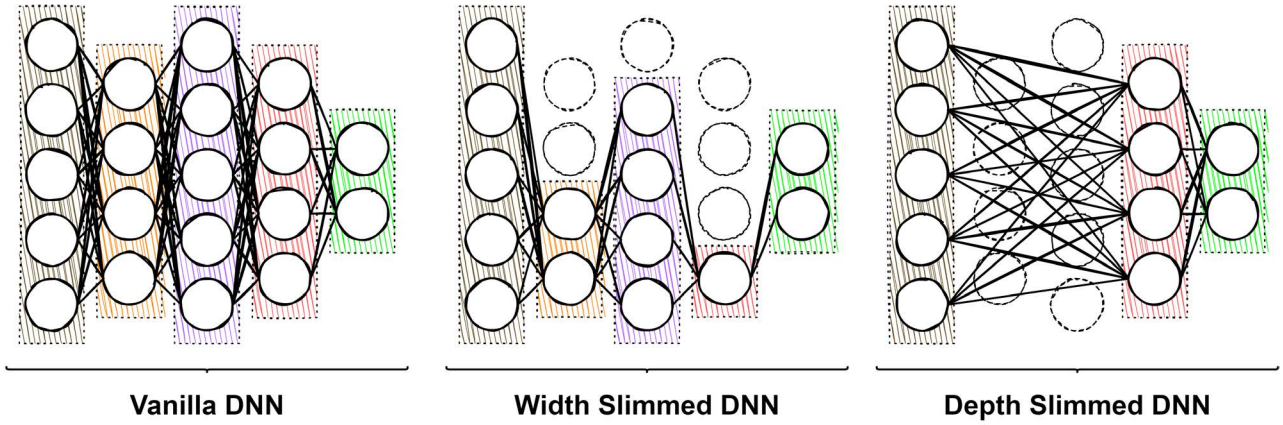

Fig. 1. From left to right: (a) the super-network, which is a vanilla DNN, (b) a sub-network activated within the super-network that has scaled down its width (number of parameters), and (c) a sub-network activated within the super-network that has scaled down its depth (number of layers).

to train a scaled down network that retains accuracy of the larger network. An ADDNN can be classified as either gated multi-branch networks, where several branches can control the computational flow while executing a neural network, or gated embedded networks, where a subset of weights are activated and shared by several embedded sub-networks within the larger super-network. The core challenge in developing an ADDNN is how to train an intelligent module to control the gates on a case-by-case basis.

II. Dynamic Deep Neural Networks

We define a DDNN primarily by the functionality to change the structure of a neural network after its construction. For the purpose of reducing the computational needs of a DNN, we focus on DDNNs that are able to scale their complexity, which takes on two forms: width and depth. See Fig 1.

A. Width Scaling

Width scaling reduces the number of parameters used in each layer of a DNN. Width parameters refer to not only the number of activation functions (*i.e.,* nodes), but also those that affect the number of computations required to execute a single layer, such as the stride and kernel size parameters used to define a convolutional layer. The most prominent type of width scaling is rooted in *slimmable* neural networks [18], [19], that create sub-networks embedded within a "super-network", where each of the sub-networks have a reduced number of active nodes. This is likened to a controlled form of dropout [20]; however, in slimmable networks, nodes are disconnected in a procedural manner rather than randomly such as they are in dropout.

B. Depth Scaling

Depth scaling reduces the number of layers used in the DNN. The most prominent type of depth scaling is early exits [21], in which the network can branch off and complete execution at an early stage of computing. Early exits provide a means to reroute the flow of computations while executing a neural network, where some branches will have fewer layers. Another powerful application of early exits is to improve the efficiency of hybrid mobile-edge computing, in what is called split computing [21], [22]. Part of the neural network is first executed on-board the mobile device. Second, supervised compression is used to reduce the dimensions of the resultant intermediate feature space. Third, the compressed data is then communicated to the edge server to finish execution of the neural network. Split computing has been shown to outperform techniques that use JPEG compression for edge computing.

C. Core Challenges

The main challenge in developing a DDNN lies within the training procedure. The neural network has to be trained to dynamically reduce the number of computations executed at runtime, while mitigating the degradation in accuracy resulting from using a less complex structure. The main solution to this challenge is knowledge distillation [12], which has been used in slimmable networks and split computing [18], [19], [22]. Knowledge distillation involves first training a teacher network, that will then teach its learned latent features to a student network. For slimmable networks, the student network is a sub-network that is embedded within the super-network (*i.e.,* having shared weights with other sub-networks) such that any of the sub-networks can be seamlessly activated at run time. For split computing, the student network is an injected bottleneck that learns a compressed representation of the teacher's latent features. The loss function used to train the student model is a linear combination of both a hard and soft target. The hard target is the desired neural network output, usually from the same loss function used to train the teacher network. The soft target is a subset of the teacher's latent features. This mitigates a degradation in accuracy, reduces the size of the student network, and teaches the student network a representation of the teacher's latent features. The key advantage of knowledge distillation, is that the teacher network is allowed to explore a high-dimensional feature space during training, while the student network learns

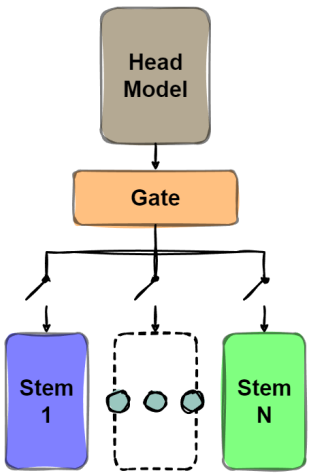

Fig. 2. A gated multi-branch network, where a switch controls which network branch to route computing into next.

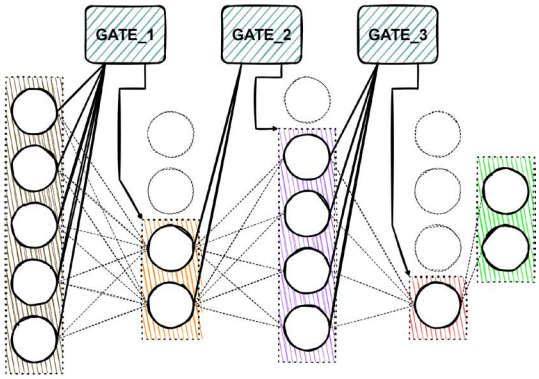

Fig. 3. A gated embedded network, where gate structures control the parameters used for sub-sets of layers within the network.

a compacted representation of this feature space to reduce the computational overhead used during execution.

III. Adaptive Dynamic Deep Neural Networks

We define an ADDNN primarily by the functionality to intelligently select how and when the structure of a neural network is changed at runtime. For the purpose of reducing the computational overhead of a DNN, we focus on ADDNNs that are used to optimize the resources utilized within a system. There are two forms of selection for neural structures: gated multi-branch networks and gated embedded sub-networks.

A. Gated Multi-Branch Networks

Gated multi-branch networks are composed of several independent, but possibly intertwined, neural networks. See Fig 2. By independent, we mean none of the weights, or other parameters, are shared between networks. By intertwined, we mean that the output of some branches may cross over into input to other branches. Each branch can be loaded into the memory for seamless switching, if the memory capacity allows it. The task of an ADDNN is then to intelligently select from the possible branches, as controlled by a gate mechanism. There are a number of such applications, in which the ADDNN controls a gated multi-branch neural network with the purpose of optimizing resource usage: BranchyNet [23] showed that an early exit model can be trained to improve both inference time and accuracy for image classification, Testudo [24] considers wireless communication resources, HydraFusion [25] considers sensor selection for object detection, Slimmable encoders [26] consider levels of quantization used in split computing, and NaviSplit [5] considers the level of compaction needed in split computing to extract depth maps from a monocular RGB image for the purpose of autonomous drone navigation.

B. Gated Embedded Networks

Gated embedded networks are composed of one super-network, with several overlapping sub-networks within said super-network. See Fig 3. By overlapping, we specify that weights of the super-network are shared within other sub-networks. The key advantages of gated embedded networks is that only the super-network, or a sufficent subset of, needs to be loaded into memory, and a sub-network and can be activated with a simple matrix multiplication by a binary indicator matrix. The first examples, to our knowledge, of an ADDNN using gated embedded networks are dynamic slimmable networks [27], [28] which control the amount of slimming (*i.e.,* percent of active nodes from the super-network) based on the difficulty of the image. Recently, NaviSlim [29] was presented to continuously change the slimming based on the perceived difficulty in a navigation task for drone autonomy. Uniquely, not only does NaviSlim control the slimming of the hidden layer nodes but also of the input layer nodes (as a means of both sensor and resolution selection).

C. Core Challenges

In both forms of gated networks, multi-branch or embedded, and across the aforementioned applications, there are some common design principles and challenges. Of most importance, is in determining the manner of intelligence used to select when and how to adjust the overall structure of, or reroute computations within, the network. This is decided by several factors, which have non-trivial relationships: (1) the mechanism used to perceive context on a case-by-case basis, (2) the resources needed to be optimized and how they correlate to that context, (3) how changing those resources will affect task accuracy, and (4) what the required level of task accuracy is as to avoid system failure and obtain mission objectives. These relationships can be summarized with the following minimization problem:

$$\begin{aligned} &\arg\min_{\phi} \ \langle R(f_\theta, g_\phi, D)\rangle \\ &\text{s.t. } \langle \eta(f_\theta, g_\phi, D)\rangle <= \beta * \langle \eta(f_\theta, D)\rangle \end{aligned} \tag{1}$$

or if reversed:

$$\begin{aligned} &\arg\min_{\phi} \ \langle \eta(f_\theta, g_\phi, D)\rangle \\ &\text{s.t. } \langle R(f_\theta, g_\phi, D)\rangle <= \beta * \langle R(f_\theta, D)\rangle \end{aligned} \tag{2}$$

where R denotes the resources expended, f_θ is the core task model (a DDNN) that is already trained, g_ϕ is some auxiliary mechanism that uses perceived context to select how to alter the structure of f_θ (making it an ADDNN), D is a provided dataset, η is the task error rate, and β is a constant. When f_θ is used without g_ϕ, then the system reverts back to a DNN, using a static amount of resources at all times – note that in the above minimization problems, we assume the DNN uses the maximum amount of resources in this case. Equation 1 minimizes the resources used while maintaining a required task accuracy. This is preferable for when one knows the mission objectives, and wants to minimize the resources expended while accomplishing them. Equation 2 minimizes the task error while forcing the resources expended to fit within an upper bound. This is preferable for when one knows the resource constraints on a limited device, and wants to obtain the best feasible performance.

The core challenge of developing an ADDNN is in designing and training g_ϕ. For example, and to this end, Hydrafusion trained a weather context classifier to control the gate of a multi-branch sensor network, NaviSplit and NaviSlim trained an auxiliary network with deep reinforcement learning to perceive the navigation difficulty and allocate resources accordingly, dynamic slimmable networks trained a classifier to determine the difficulty of an input image, and BranchyNet measured the entropy level at several layers to determine when to early exit based on a learned threshold. Some key difficulties in training g_ϕ are:

1) If training both sets of parameters, ϕ and θ, in a dual-optimization problem, g_ϕ is attempting to learn how to alter the structure of f_θ while the parameters of f_θ are changing. This can create instabilities while updating ϕ and θ during their optimization, by creating a game of tug-of-war where parameters oscillate around eachother. The solution to this in literature, has been to decouple the training processes of g_ϕ and f_θ.
2) The high level latent features realized within the parameters, θ, may not only be useful for the task that f_θ is being trained for, but also when using g_ϕ for selecting the structure of f_θ. The paradox observed here, is that the high level features have already been computed in f_θ and thus if g_ϕ is to use them as inputs then there will be diminishing, if not at times unreachable, resource savings. A unique solution to this in literature [5], [29], is to consider the data as timeseries so that the next data instance is highly correlated to the current one. This way, the high level features computed in f_θ can be used by g_ϕ to allocate proper resources in the next timestep.
3) If using deep reinforcement learning, training convergence times can be on the magnitude of months [30]. The training time is further exacerbated by how sensitive deep reinforcement learning algorithms are to selected hyper parameters, requiring a large grid to be explored.

IV. Conclusion

We have demonstrated the evolution of a DNN, which was originally static in nature with a fixed computational overhead set at runtime. An evolution to this is a DDNN, which allows the structure of a DNN to become dynamic so that the computational overhead can change at runtime. The State-of-the-Art evolution is an ADDNN, which adds the functionality to intelligently select how the DDNN should change its structure to better adapt to context, and fulfill accuracy and/or resource constraints. The difficulties of training a DDNN lie within training the task model to have a smaller computational overhead while mitigating performance degradation. The difficulties of training an ADDNN lie within the non-trivial dichotomy between the two systems of the task model, used to fulfill mission objectives, and the auxiliary module, used to adapt the computational overhead to perceived context.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[2] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural computation*, vol. 29, no. 9, pp. 2352–2449, 2017.

[3] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," *Proceedings of the IEEE*, vol. 111, no. 3, pp. 257–276, 2023.

[4] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos, "Image segmentation using deep learning: A survey," *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 7, pp. 3523–3542, 2021.

[5] T. K. Johnsen, I. Harshbarger, Z. Xia, and M. Levorato, "Navisplit: Dynamic multi-branch split dnns for efficient distributed autonomous navigation," *arXiv preprint arXiv:2406.13086*, 2024.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[7] A. Pappalardo, Y. Umuroglu, M. Blott, J. Mitrevski, B. Hawks, N. Tran, V. Loncar, S. Summers, H. Borras, J. Muhizi, M. Trahms, S.-C. Hsu, S. Hauck, and J. Duarte, "Qonnx: Representing arbitrary-precision quantized neural networks," 2022.

[8] R. Banner, I. Hubara, E. Hoffer, and D. Soudry, "Scalable methods for 8-bit training of neural networks," 2018.

[9] M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplications," 2015.

[10] A. Gholami, K. Kwon, B. Wu, Z. Tai, X. Yue, P. Jin, S. Zhao, and K. Keutzer, "Squeezenext: Hardware-aware neural network design," 2018.

[11] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning?" *Proceedings of machine learning and systems*, vol. 2, pp. 129–146, 2020.

[12] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[13] S. Ahn, S. X. Hu, A. Damianou, N. D. Lawrence, and Z. Dai, "Variational information distillation for knowledge transfer," 2019.

[14] A. Mishra and D. Marr, "Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy," 2017.

[15] V. Vanhoucke, A. Senior, M. Z. Mao *et al.*, "Improving the speed of neural networks on cpus," in *Proc. deep learning and unsupervised feature learning NIPS workshop*, vol. 1, no. 2011, 2011, p. 4.

[16] M. Sapienza, E. Guardo, M. Cavallo, G. Torre, G. Leombruno, and O. Tomarchio, "Solving critical events through mobile edge computing: An approach for smart cities," 05 2016, pp. 1–5.

[17] C.-C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose, "Videoedge: Processing camera streams using hierarchical clusters," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, 2018, pp. 115–131.

[18] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, "Slimmable neural networks," *arXiv preprint arXiv:1812.08928*, 2018.

[19] J. Yu and T. S. Huang, "Universally slimmable networks and improved training techniques," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1803–1811.

[20] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[21] Y. Matsubara, M. Levorato, and F. Restuccia, "Split computing and early exiting for deep learning applications: Survey and research challenges," *ACM Computing Surveys*, vol. 55, no. 5, pp. 1–30, 2022.

[22] Y. Matsubara, D. Callegaro, S. Singh, M. Levorato, and F. Restuccia, "Bottlefit: Learning compressed representations in deep neural networks for effective and efficient split computing," in *2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. IEEE, 2022, pp. 337–346.

[23] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *2016 23rd international conference on pattern recognition (ICPR)*. IEEE, 2016, pp. 2464–2469.

[24] M. Odema, L. Chen, M. Levorato, and M. A. Al Faruque, "Testudo: Collaborative intelligence for latency-critical autonomous systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 6, pp. 1770–1783, 2022.

[25] A. V. Malawade, T. Mortlock, and M. A. Al Faruque, "Hydrafusion: Context-aware selective sensor fusion for robust and efficient autonomous vehicle perception," in *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2022, pp. 68–79.

[26] J. S. Assine, E. Valle, M. Levorato *et al.*, "Slimmable encoders for flexible split dnns in bandwidth and resource constrained iot systems," *arXiv preprint arXiv:2306.12691*, 2023.

[27] C. Li, G. Wang, B. Wang, X. Liang, Z. Li, and X. Chang, "Dynamic slimmable network," in *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, 2021, pp. 8607–8617.

[28] Z. Jiang, C. Li, X. Chang, L. Chen, J. Zhu, and Y. Yang, "Dynamic slimmable denoising network," *IEEE Transactions on Image Processing*, vol. 32, pp. 1583–1598, 2023.

[29] T. K. Johnsen and M. Levorato, "Navislim: Adaptive context-aware navigation and sensing via dynamic slimmable networks," in *2024 IEEE/ACM Ninth International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE, 2024, pp. 110–121.

[30] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.