Octopus: Context-Aware CNN Inference for IoT Applications

Mohammad Motamedi, Felix Portillo, Mahya Saffarpour, Daniel Fong, and Soheil Ghiasi
Electrical and Computer Engineering Department, University of California, Davis, USA.

***Abstract*—Modern Convolutional Neural Networks (CNNs) in computer vision are trained on a large number of images from numerous categories to form rich discriminative feature extractors. Inference using such models on resource-constrained IoT platforms poses a challenge and an opportunity: Having limited computation, storage and energy budgets, most IoT platforms are not capable of hosting such compute intensive models. However, typical IoT applications demand detection of a relatively small number of categories, albeit the specific categories of interest may change at runtime as the context evolves dynamically. In this paper, we take advantage of the opportunity to address the challenge. Specifically, we develop a novel transformation to the architecture of a given CNN, so that the majority of the inference workload is allocated to class-specific disjoint branches, which can be dynamically executed or skipped, based on the context, to fulfill the application requirements. Experiments demonstrate that our approach preserves the classification accuracy for the classes of interest, while proportionally decreasing the model complexity and inference workload.**

***Index Terms*—Convolutional Neural Networks, Embedded Systems, Software Synthesis.**

I. Introduction

The current practice in designing CNNs does not offer class-based parameter allocation. Therefore, different parameters that contribute to recognizing members of each class are distributed in sporadic kernels across the CNN. As a result, if we intend to eliminate a subset of classes in a trained CNN, parameters that correspond to these classes can be neither identified nor deleted. In this paper, we propose a systematic approach for distinguishing those parameters in a trained CNN, that help it understand *images of each class*. To further elaborate, let us assume a trained CNN, Ψ, can classify a set of α categories, A, where $\alpha = |A|$. Given an IoT application that is interested in recognition of members of set B such that $B \subset A$, $\beta = |B|$, and $\beta \ll \alpha$, the goal is to derive a new CNN, Ψ', from Ψ under the following conditions: First, Ψ' should be capable of classifying images of set B. Therefore, it utilizes a β-way classifier rather than the α-way classifier that is used in Ψ. Second, The classification accuracy of Ψ' should be equal to that of Ψ.

Ideally, Ψ' should not include any of the parameters whose exclusive role is to help Ψ in recognizing images that belong to members of $\bar{B} = A - B$. In other words, those kernels which contribute to extracting features that are exclusively used by Ψ for recognizing members of $\bar{B}$ should no longer exist in Ψ'. Furthermore, An ideal solution should make it possible to derive Ψ' from Ψ without any further training or fine-tuning. Moreover, it is desirable to have the possibility of performing a reverse mapping in which we can restore Ψ from Ψ' with no additional computations. Such an option makes it possible for the target IoT application to change its functionality on the fly in a context-aware fashion. That is, if the context changes, the application should be able to reconfigure the CNN.

When a CNN is designed to recognize α image classes, it is reasonable to conclude that on average approximately $1/\alpha$ of total parameter budget is dedicated to learn members of each class. Hence, for an ideal case, Equation (1) yields the class-wise parameter budget in Ψ, and Equation (2) computes a lower bound for the expected number of parameters in Ψ'. In this paper, we use Γ to refer to the average parameter budget that a CNN requires for recognizing members of each class.

$$\Gamma_{\Psi}^{Ideal} = \frac{\text{Num. of Params. in } \Psi}{\alpha} \tag{1}$$

$$\text{Num. of Params. in } \Psi' \leq \text{Num. of Params. in } \Psi \times \frac{\beta}{\alpha} \tag{2}$$

It is worth noting that class-agnostic kernels are used by multiple classes. Such a sharing increases the quality of feature extraction since each class can take advantage of more than $1/\alpha$ of parameters. However, it decreases the number of class-specific kernels. As a results, less than $1/\alpha$ of the total parameter budget is used exclusively for understanding images that belong to each class. On this ground, it is not feasible to achieve a linear parameter scale-down while deriving Ψ' from Ψ. The term "parameter budget" is used in this article to refer to the "number of parameters" in a CNN.

In this paper, we present a new CNN architecture, Octopus, which is designed with the idea of having a disjoint set of parameters for distinct image categories. We use this architecture to map a state-of-the-art model which is trained on the ILSVRC dataset [1] to a smaller network that is favorable for an application-specific embedded platform, and measure the parameter reduction ratio. We show that scaling down a CNN with the Octopus architecture to a smaller CNN requires no additional computation, yields a very high parameter reduction, and incurs no accuracy loss.

The research community has put forth numerous approaches for improving the execution time and power efficiency of CNNs on embedded platforms. Designing customized hardware accelerators, mobile SoC-oriented CNN optimizations, pruning, quantization, distillation, and compact model design are some of the most important approaches that are used to address the high computational demands of CNNs [2].

While we target embedded platforms, our work is substantially different from the aforementioned endeavors. The key philosophy behind the proposed approach in this work is elimination of parameters which exclusively participate in understanding images that fall out of the scope of interest of a particular IoT application. This is an orthogonal dimension to the general network optimization approaches where essential parameters for recognition of members of all classes remain unaffected. As a result, a CNN which is pruned, quantized, or compressed can be used as the base network, Ψ, in the $\Psi \rightarrow \Psi'$ mapping. In our experiments, we use GoogLeNet [3], which has a compressed architecture due to use of reduce-expand layers, as our baseline CNN.

To the best-of-our-knowledge, two other work address the problem of mission-driven CNN optimization: Distill-Net [4]

and Resource-Scalable CNN Synthesis (RSCS) [5]. The former work analyzes the forward path to identify neurons that are dormant during recognition of images that belong to the target application's scope of interest. Subsequently, all such neurons will be eliminated from the CNN. The latter work, offers an automated approach for synthesizing a CNN architecture that is tailored to the needs of a given IoT-grade task. We compare this work with Distill-Net [4] and RSCS [5] in our experiments.

II. Octopus Architecture

In the Octopus architecture, those parameters of the CNN that are used exclusively for understanding members of each class will be located in distinct, class-specific components. Ideally, such components should be dense, and loosely-connected to the body of the neural network.

Impact of Dense Components: The main objective of performing the $\Psi \rightarrow \Psi'$ mapping is to reduce the required computations proportionally to the complexity of the target IoT-grade application. However, since the current computational infrastructures do not perform well on sparse data structures, we aim to ensure Ψ' has a dense parameter representation.

Impact of Loose Connections: We require the aforementioned class-specific components to be loosely connected to the body of the neural network. That is, first, detaching them shall not demand any further training or fine-tuning. Second, the functionality of such components should be independent of each other. Hence, removing one or more of them should not adversely affect the quality of feature extraction in others.

The problem of mission-driven CNN optimization can be drastically simplified using a CNN architecture analogous to the one outlined above. To further clarify, in such an architecture if an image class is no longer beneficial for a given IoT-grade task, we can simply remove it along with its corresponding class-specific component[1].

Let us assume that we want to create a CNN with a restricted parameter budget of Φ, to classify images of a dataset that contains α classes. Based on the current literature, one should conclude that a conventional architecture, such as those proposed in [3], [6], would suffice for such a task. The question is: How α distinct CNNs, with a parameter budget of Φ/α for each of them, would perform? We hypothesize that such an architecture would suffer from kernel duplication. In CNNs, features that are extracted from a kernel, specifically in shallow layers, can be utilized for understanding more than one category of images. However, in the suggested architecture, all paths are disjoint, and thus a shared utilization of kernels is infeasible. Hence, each small CNN will have to create a private copy of the same kernels with analogous feature extraction signatures. Given the fixed parameter budget, such a phenomenon hinders the CNN in forming a wide variety of different kernels whose collaborative effort is essential for effective recognition of different images.

The relation of the extracted features in shallow layers to the input is mostly structural, however, that of deep layers tend to be mainly semantical [7]. In a CNN, classification judgments

[1]In practice, we never remove a class-specific component. We simply prevent the data from flowing through it. Hence, it gets neither loaded nor computed.

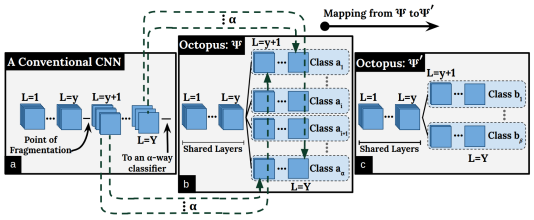

Fig. 1. The Octopus architecture. Class-specific kernels are placed in edge-disjoint components to make it possible to efficiently omit them when their corresponding classes are not required. Part (a) shows a conventional CNN with Y layers that is designed to recognize α classes. Part (b) shows the Octopus equivalent (Y layers, α classes) of the same architecture where path to each class is exclusive after the point of fragmentation. Part (c) illustrates a mapping from Ψ to Ψ', where the target IoT application requires to classify members of set B with β classes: $B = \{b_1, \cdots, b_\beta\}$.

are based on the features that are extracted from the deepest layer, since they have the highest degree of distinctiveness (i.e., highest class-specificity). This characteristic deteriorates as we move towards the shallowest layer. On this ground, we hypothesize that kernels of deep layers can be placed in class-specific edge-disjoint components, while kernels of shallow layers should be jointly used in understanding all classes.

Consequently, we propose the Octopus architecture which is illustrated in Figure 1. The first y layers are stacked in the conventional fashion, in that the extracted features from these layers are used for understanding images of all classes. Starting from layer $(y+1)$ the path for each class is exclusive. The consecutive kernels in each class-specific path will be trained to extract features that are only beneficial for their class. Class-specific components that start from layer $(y+1)$ are completely disjoint. This flexibility allows a particular embedded application to easily omit computations that correspond to a class that can be ignored. In Figure 1, the original network (shown in part a) has Y layers and is designed to classify α classes. The Octopus counterpart of that network (shown in part b), has identical specifications. It includes Y layers and can classify α classes. The difference is that starting from layer y, it offers distinct data path for each target class. Finally, the output of a $\Psi \rightarrow \Psi'$ mapping is illustrated in part (c). In this particular mapping, Ψ is tailored to the demands of an IoT-grade application that is interested in understanding β classes, $B = \{b_1, \cdots, b_\beta\}$.

In the process of deriving Ψ' from Ψ, data paths that are exclusive to the members of set $\bar{B}$ will be disconnected. The resulting network, Ψ', simply uses the class-specific components that correspond to the images of set B. Since the extraneous data routes are disconnected in Ψ', the network no longer needs to load their corresponding parameters or perform their computations.

We develop Octopus to be compatible and complementary to the existing CNN architectures. In what follows, we present an approach for deriving the Octopus equivalent of a given CNN architecture under a constrained parameter budget of Φ. Let us assume the original architecture has N convolutional layers, and the parameter budget for each of which is shown by Φ_n such that Equation (3) holds.

$$\Phi = \sum_{n=1}^{N} \Phi_n \tag{3}$$

Since we use an optimized state-of-the-art neural network for the baseline CNN, it is desirable to preserve its ability in extracting high quality features. Hence, in synthesizing Octopus counterpart of a convnet, we refrain from changing the type, size, and order of the convolution kernels in each layer. However, the numbers of filter-banks that are placed after the point of fragmentation is subject to change. The point of fragmentation is the layer at which the class-specific paths begin. All filter-banks have the same dimensions, share the same input, and they yield structurally analogous features. Such a redundancy is essential for extracting a plethora of features that are required for understanding different classes. However, each class-specific component is only responsible for understanding members of one class. As a result, in developing the Octopus counterpart of a CNN, decreasing the number of filter-banks in class-specific components is a reasonable choice.

Assuming that the original CNN is designed for classifying α classes, Equation (4) computes the number of filter-banks in layer n of each class-specific component. In this equation, K_n and Φ_n are the kernel size, and the number of parameters in layer n, respectively. Also, the number of Input Feature Maps is shown by #IFMs. This equation is derived for square kernels and requires minor adjustments before being used for asymmetric kernels.

$$\#\text{ Filter-banks} = \begin{cases} \lceil \frac{\Phi_n}{\alpha \times \#\text{IFMs}_n \times K_n^2} \rceil, & \text{if } n > y \\ \lceil \frac{\Phi_n}{\#\text{IFMs}_n \times K_n^2} \rceil, & \text{otherwise} \end{cases} \quad (4)$$

Having bottleneck in a CNN hinders its ability in transferring extracted features to the subsequent layers. As a result, the quality of achieved features in the last layer deteriorates. To prevent this, in synthesizing the Octopus counterpart for a given CNN, we need to ensure that Equation (5) holds. This equation implies that a bottleneck happens when the aggregated capacity of class-specific components is smaller than that of the previous layer in the CNN.

$$M_n \leq \begin{cases} \alpha \times M_{n+1}, & \text{if } n = y \\ M_{n+1}, & \text{otherwise} \end{cases} \quad (5)$$

III. Results and Discussions

A. Case Study: Impact of Point of Fragmentation

In this section, we study the effect of the location of fragmentation on CNNs learning ability. We use the Inception model [3] as our baseline neural network and create a new CNN with a capacity of 99,000 parameters for classifying the CIFAR-10 dataset. The capacity of this network is approximately 1.4% of GoogLeNet, and its number of classes is 1% of the number of categories that GoogLeNet is designed to classify. In our experiments, we move the point of fragmentation from the first layer to the last layer and study the impact of its location on the classification accuracy. The Inception-based Octopus architectures that are used for these experiments are detailed in Table I. In this table, the underlined values indicate that their corresponding layers are shared among all classes. For instance, the first value of the second row is underlined. Therefore, in the corresponding architecture, the first layer (i.e., Conv 1) is shared. In the same row (row #2), other values are not underlined. That is, layers which correspond to them are not shared. If a number is not underlined, it represents a per branch parameter budget. As an example, in Ci10 B, each branch has its own Conv 2 with 75 parameters. Therefore, as the CNN has 10 classes, the total number of parameters for that layer equals to 75×10.

The baseline model (the last row in Table I) includes no class-specific branch (i.e., has a conventional architecture), and requires 140 epochs to reach its maximum accuracy (86%). For comparison, we report the accuracy of other models after 140 epochs of training. In our experiments, we used stochastic gradient descent using adaptive momentum estimation [9] with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and a learning rate of 0.001. We utilized data augmentation including image translation, rotation, and horizontal flipping. The process is performed on images randomly selected from each batch with varying degrees of translation, and rotation.

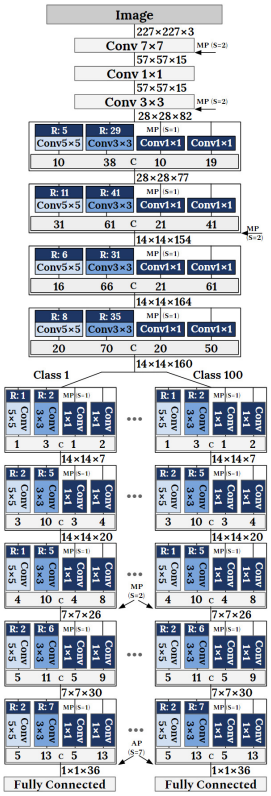

Fig. 2. GoogLeNet [3] architecture in Octopus fashion designed to classify 100 randomly selected image classes from the ImageNet 2012 competition dataset [8]. In this figure, MP stands for Max Pooling layers, and R: x is a 1×1 convolutional layer which includes x channels.

As Table I presents, moving the point of fragmentation towards deep layers increases the accuracy achieved in 140 epochs. However, the trend saturates when adequate number of layers are shared. When the number of shared layers is very small, each branch has to create its own copy of those kernels that otherwise could have been shared. Given the fixed parameter budget, this prevents the CNN from forming a variety of kernels whose existence is essential for a better performance.

B. CNN Design Based on ILSVRC Dataset

We used the Inception [3] model as a baseline architecture and created an Octopus-based neural network (i.e., Ψ) capable of classifying 100 image classes. As Figure 2 illustrates, the newly created architecture has 100 class-specific branches, each capable of understanding images that belong to a target class. The CNN is trained using the hyper-parameters that are mentioned in Subsection III-A and benefits from the same data augmentation approach. Suggested by our case study, the point of fragmentation is selected to be after the layer I4B.

As Equation (6) demonstrates, a simple Octopus-based CNN, such as the one presented in Figure 2, is powerful enough to

TABLE I

CNNs WITH DIFFERENT POINTS OF FRAGMENTATION, SIGNIFIED BY AN ASTERISK. TO STUDY ITS IMPACT ON ACCURACY, WE MOVED THE FRAGMENTATION POINT FROM THE INPUT LAYER (IN CNN NAMED CI10 A) TO LAYER I4B (IN CNN NAMED CI10 H). THE UNDERLINED VALUES SHOW NUMBERS OF PARAMETERS IN SHARED LAYERS. OTHER VALUES, INDICATE THE BRANCH-SPECIFIC PARAMETER BUDGETS OF THEIR CORRESPONDING LAYERS.

CNN Name	Number of Parameters													Frag. Point	Num. Shared Layers	Total Params. (K)	Accuracy (%)
	Conv 1	Conv 2	Conv 3	I3A	I3B	I4A	I4B	I4C	I4D	I4E	I5A	I5B	FC				
Ci10 A	441	9	162	115	368	377	377	690	880	910	1416	1879	2304	Input	0 / 49	99	75
Ci10 B	*3675	75	162	115	368	377	377	690	880	910	1416	1879	2304	Conv 1	1 / 49	99	77
Ci10 C	1323	*81	486	115	368	377	377	690	880	910	1416	1879	2304	Conv 2	2 / 49	99	77
Ci10 D	2499	289	*2754	187	368	377	377	690	880	910	1416	1879	2304	Conv 3	3 / 49	99	81
Ci10 E	2499	289	2601	*1006	511	377	377	690	880	910	1416	1879	2304	3A	8 / 49	99	83
Ci10 F	2499	289	2601	1006	*2345	608	377	690	880	910	1416	1879	2304	3B	13 / 49	99	83
Ci10 G	2499	289	2601	1006	2345	*2648	696	690	880	910	1416	1879	2304	4A	18 / 49	99	84
Ci10 H	2499	289	2601	1006	2345	2648	*3374	1096	880	910	1416	1879	2304	4B	23 / 49	99	86
Ci10 I	2499	289	2601	1006	2345	2648	3374	3699	4362	5367	6222	9740	55K	N/A	49 / 49	99	86

TABLE II

PERFORMANCE COMPARISON BETWEEN DIFFERENT APPROACHES FOR MISSION-DRIVEN CNN SYNTHESIS TARGETING IOT-GRADE APPLICATIONS.

# Cls. in Target IoT-grade CNN	Number of Parameters (M)†			Num. Params: Achieved/Ideal (%)†§			GFLOPS†			Classification Accuracy‡			Inference Time (ms)†			Speedup to GoogLeNet‡		
	Distil.	RSCS	Octopus	Distil.	RSCS	Octopus	Distil.	RSCS	Octopus	Distil.	RSCS	Octopus	Distil.	RSCS	Octopus	Distil.	RSCS	Octopus
5	2.61	1.77	0.18	87.00X	59.00X	6.00X	2.20	1.39	0.26	68	85	68	990	671	68	2.68X	3.95X	38.83X
10	3.58	1.80	0.21	51.14X	25.71X	3.00X	2.42	1.40	0.27	68	83	68	1358	683	80	1.95X	3.88X	33.29X
15	3.97	1.82	0.24	39.70X	18.20X	2.40X	2.56	1.42	0.27	68	84	68	1506	690	91	1.76X	3.84X	29.13X
20	4.55	1.85	0.27	32.50X	13.21X	1.93X	2.78	1.46	0.28	68	81	68	1726	702	102	1.54X	3.78X	25.89X
25	4.90	1.88	0.30	28.82X	11.06X	1.76X	2.89	1.47	0.29	68	82	68	1858	713	114	1.43X	3.72X	23.30X
30	5.02	1.90	0.33	23.90X	9.05X	1.57X	2.92	1.49	0.29	68	84	68	1904	721	137	1.39X	3.68X	21.18X
Average	4.11	1.84	0.26	43.84X	22.71X	2.78X	2.63	1.44	0.28	68	83.17	68	1557	696.67	96.67	1.79X	3.81X	28.60X

† Smaller is better. ‡ Larger is better. § The ideal value is 1X.

TABLE III

COMPARISON OF DEPLOYMENT COMPLEXITY FOR DIFFERENT APPROACHES.

Approach	Profiling	Training	Reverse Mapping	Dynamic Context-aware Reconfiguration
Distill-Net	Yes	No	Impossible	Impossible
RSCS	No	Yes	Impossible	Impossible
Octopus	No	No	Possible	Possible

be used for dynamic creation of 1.27×10^{30} different sub-architectures without any further training or fine-tuning.

$$2^{100} - 1 = \sum_{i=1}^{100} \binom{100}{i} \tag{6}$$

Using the architecture presented in Figure 2, we created five different CNNs, and compared their performances with Distill-Net [4] and RSCS [5]. Table II presents the comparison results. Among all of the different approaches, RSCS in the only one that demands a fresh CNNs training for each new task (i.e., a whole new training procedure for each record of Table II). Unsurprisingly, this approach yields the highest accuracy.

Compared to other approaches, Octopus yields the best model size with an average parameter budget of 0.26 Million elements. This value is 7 times better than the second best approach and it is only 2.78 times larger than the *ideal* model size (Equation (2)). Octopus also outperforms other approaches in terms of the required computational budget (5.14 times better than the second best approach).

The inference time for all of the CNNs is approximated targeting the Qualcomm Snapdragon 800 SoC, using the approach proposed in [10], and the results which are presented in Table II demonstrate that the proposed approach requires 97 (ms) on average for each inference. This value is considerably better than 697 (ms) that RSCS-based CNNs demand, and drastically better than 1557 (ms) that Distill-Net-based CNNs require.

As Table III presents, for each of the CNNs in Table II, Distill-Net requires to profile the baseline neural network, Ψ, to identify and eliminate the extraneous parameters. Analogously, for each record of this table, RSCS demands a whole new training procedure. Meanwhile, the proposed approach demands absolutely no computations for creating each of the presented CNNs. In Octopus, the process of $\Psi \rightarrow \Psi'$ mapping can be performed dynamically, simply by controlling the data flow in the CNN. Finally, the process of deriving Ψ' in an Octopus-based architecture is not destructive. As a result, it is possible to perform a reverse mapping to retrieve the original neural network. This mapping offers dynamic context-aware CNN reconfiguration since Ψ can be reduced to a newer mission-specific CNN, Ψ''.

REFERENCES

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 248–255.

[2] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich *et al.*, "Going deeper with convolutions." Cvpr, 2015.

[4] M. Motamedi, F. Portillo, D. Fong, and S. Ghiasi, "Distill-net: Application-specific distillation of deep convolutional neural networks for resource-constrained iot platforms," *arXiv preprint arXiv:1812.07390*, 2018.

[5] M. Motamedi, F. Portillo, M. Saffarpour, D. Fong, and S. Ghiasi, "Resource-scalable cnn synthesis for iot applications," *arXiv preprint arXiv:1901.00738*, 2019.

[6] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826.

[7] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[8] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[9] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[10] M. Motamedi, D. Fong, and S. Ghiasi, "Cappuccino: Efficient cnn inference software synthesis for mobile system-on-chips," *IEEE Embedded Systems Letters*, 2018.