Incomplete Dot Products for Dynamic Computation Scaling in Neural Network Inference

Bradley McDanel
Harvard University
Cambridge, MA, USA
Email: mcdanel@fas.harvard.edu

Surat Teerapittayanon
Harvard University
Cambridge, MA, USA
Email: steerapi@seas.harvard.edu

H.T. Kung
Harvard University
Cambridge, MA, USA
Email: kung@harvard.edu

***Abstract*—We propose the use of incomplete dot products (IDP) to dynamically adjust the number of input channels used in each layer of a convolutional neural network during feedforward inference. IDP adds monotonically non-increasing coefficients, referred to as a "profile", to the channels during training. The profile orders the contribution of each channel in non-increasing order. At inference time, the number of channels used can be dynamically adjusted to trade off accuracy for lowered power consumption and reduced latency by selecting only a beginning subset of channels. This approach allows for a single network to dynamically scale over a computation range, as opposed to training and deploying multiple networks to support different levels of computation scaling. Additionally, we extend the notion to multiple profiles, each optimized for some specific range of computation scaling. We present experiments on the computation and accuracy trade-offs of IDP for popular image classification models and datasets. We demonstrate that, for MNIST and CIFAR-10, IDP reduces computation significantly, e.g., by 75%, without significantly compromising accuracy. We argue that IDP provides a convenient and effective means for devices to lower computation costs dynamically to reflect the current computation budget of the system. For example, VGG-16 with 50% IDP (using only the first 50% of channels) achieves 70% in accuracy on the CIFAR-10 dataset compared to the standard network which achieves only 35% accuracy when using the reduced channel set.**

I. Introduction

Inference with deep Convolutional Neural Networks (CNNs) on end or edge devices has received increasing attention as more applications begin to use sensor data as input for models running directly on the device (see, e.g., [1]). However, each trained CNN model has a fixed accuracy, size, and latency profile determined by the number of layers and parameters in each layer. The static nature of these models can be problematic in dynamic contexts, such as when running on mobile device, where the power and latency requirements of CNN inference may change based on the current battery life or computation latency allowance of the device.

One approach to address these dynamic contexts is to train multiple CNN models of ranging sizes, such as by varying the number of parameters in each layer as in MobileNet [2], and then selecting the appropriate model based on the current system requirements. However, this kind of approach requires storing multiple models of different sizes to support the desired computation flexibilities of the system. Ideally, we would instead train a single CNN that could dynamically scale across a computation range in high resolution, trading off accuracy for power consumption and latency as desired.

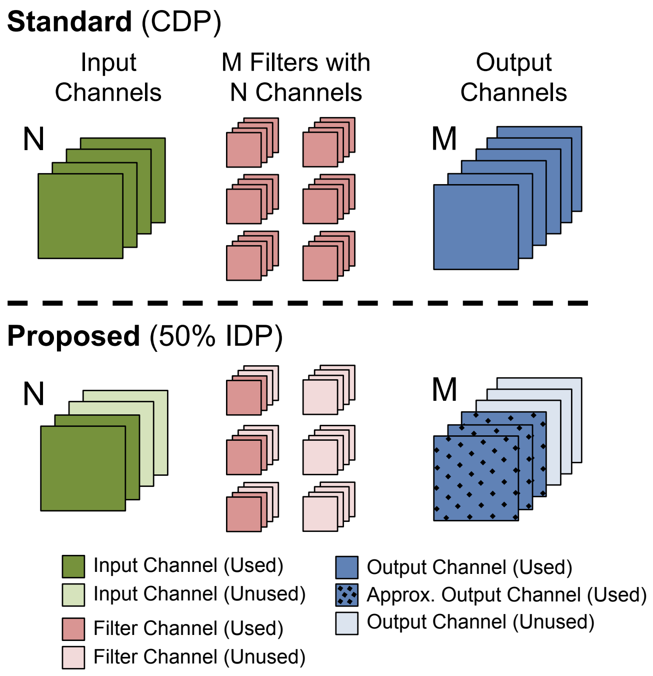

Fig. 1: Contrasting computation using complete dot product (CDP) in standard networks and incomplete dot product (IDP) proposed in this paper for a convolutional layer. Under standard CNN, for each filter at a given layer, N input channels are used in the dot product computation (CDP), to compute the corresponding output channel. Under, for example, 50% IDP, the filter uses the first 50% of the input channels to compute the corresponding output channel, which is an approximation of the CDP. Furthermore, only 50% of these filters are used since output channels for the other filters will not be utilized in the next layer. This leads to a 75% reduction in computation for 50% IDP.

To provide a solution for dynamic scaling over a computation range, we propose to modify CNN training, by adding a *profile* of monotonically non-increasing channel coefficients for channels in a given layer. These coefficients provide an ordering for channels to allow channels with small coefficients to be dropped out during inference to save computation. By training a CNN with this profile, dot products performed during forward propagation may use only some of the beginning channels in each filter, without compromising accuracy significantly. We call such a dot product *incomplete dot product* (IDP). As we show in our evaluation in Section IV, using IDP on networks trained with a properly chosen profile achieves much higher accuracy relative to using IDP on standard CNNs (trained without channel coefficients). Using the IDP scheme, we are able to train a single CNN that can dynamically scale across a computation range.

Figure 1 contrasts forward propagation in a standard CNN layer using all input channels, which we refer to as *complete dot product* (CDP), to that of using only half of the input channels (50% IDP). IDP has two sources of computation reduction: 1) each output channel is computed with a filter using only 50% of the input channels (meaning the output is an approximation of using 100% of channels) and 2) half of the filters are unused since their corresponding output channels are never consumed in the next layer. Therefore, 50% IDP leads to a 75% reduction in computation cost. In general, the IDP computation cost is p^2 times the original computation cost, where p is the percentage of channels used in IDP. As we show in our analysis in Section IV, IDP reduces computations (number of active input channels used in forward propagation) of several conventional network structures while still maintaining similar accuracy when all channels are used.

In the next section, we describe how IDP is integrated into multiple layer types (fully connected, convolution, and depthwise separable convolution) in order to train such networks where only a subset of channels may be used during forward propagation. We call such networks incomplete neural networks.

II. Incomplete Neural Networks

An incomplete neural network consists of one or more incomplete layers. An incomplete layer is similar to a standard neural network layer except that incomplete dot product (IDP) operations replace conventional complete dot product (CDP) operations during forward propagation. We begin this section by describing IDP during forward propagation and provide an explanation of how IDP is specifically added to fully-connected, convolutional, and depthwise separable convolutional layers.

A. Incomplete Dot Product

IDP adds a profile consisting of monotonically non-increasing coefficients $\boldsymbol{\gamma}$ to the components of the dot product computations of forward propagation. These coefficients order the importance of the components in decreasing order from the most to least important. Mathematically, the incomplete dot product (IDP) of two vectors $(a_1, a_2, \dots, a_N)^\intercal$

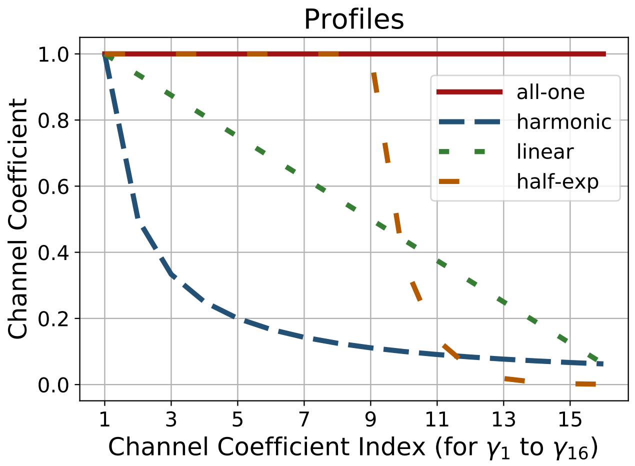

Fig. 2: The profiles evaluated in this paper. Each channel coefficient index on the x-axis corresponds to an input channel (as shown in Figures 3 and 4). The y-axis shows the value of each coefficient for a profile. The all-one (all coefficients equal 1) profile corresponds to a standard network layer without channel coefficients. The other profiles provide different schemes for the coefficients. For instance, in half-exp, an exponential decay for second half of coefficients is used.

and $(b_1, b_2, \dots, b_N)^\intercal$ is a truncated version of the following expression

$$\sum_{i=1}^{N} \gamma_i a_i b_i = \gamma_1 a_1 b_1 + \gamma_2 a_2 b_2 + \cdots + \gamma_N a_N b_N$$

which keeps only some number of the beginning terms, where $\gamma_1, \gamma_2, \dots, \gamma_N$ are the monotonically non-increasing profile coefficients.

To compute IDP with a target IDP percentage, the beginning components, starting with $\gamma_1 a_1 b_1$, are accumulated until the target percentage of components is reached. The contribution of the remaining components, with smaller coefficients, is ignored, making the dot product incomplete. This is mathematically equivalent to setting the unused coefficients to 0.

B. Choice of Profile

There are multiple ways to set the profile $\boldsymbol{\gamma} = \{\gamma_1, \gamma_2, \dots, \gamma_i, \dots, \gamma_N\}$ corresponding to various policies which may favor dynamic computation scaling for certain ranges at the expense of other regions. In Section IV-C, we discuss the performance implications of the various profiles. The profiles evaluated in this paper, shown in Figure 2, are:

1) all-one: corresponds to a standard network layer without a profile

$$\gamma_i = 1 \text{ for } i = 1, \dots, N$$

2) harmonic: a harmonic decay

$$\gamma_i = \frac{1}{i} \text{ for } i = 1, \dots, N$$

3) linear: a linear decay

$$\gamma_i = 1 - \frac{i}{N} \text{ for } i = 1, \dots, N$$

4) half-exp: all-one for the first half of terms and an exponential decay for latter half of terms

$$\gamma_i = \begin{cases} 1, & \text{if } i < \frac{N}{2} \\ \exp\left(\frac{N}{2} - i - 1\right), & \text{otherwise} \end{cases} \text{ for } i = 1, \dots, N$$

C. Incomplete Layers

In this section, we describe how IDP is integrated into standard neural network layers, which we refer to as incomplete layers.

1) Incomplete Fully Connected Layer: A standard linear layer does the following computation:

$$y_j = \sum_{i=1}^{N} w_{ji} x_i,$$

where $j \in \{1, \dots, M\}$, M is the number of output components, N is the number of input components, w_{ji} is the layer weight corresponding to the j-th output component and i-th input component, x_i is the i-th input component and y_j is j-th output component. An incomplete linear layer does the following computation instead:

$$y_j = \sum_{i=1}^{N} \gamma_i w_{ji} x_i,$$

where γ_i is the i-th profile coefficient.

2) Incomplete Convolution Layer: A standard convolution layer does the following computation:

$$\mathbf{y}_j = \sum_{i=1}^{N} \mathbf{f}_{ji} * \mathbf{x}_i,$$

where $j \in \{1, \dots, M\}$, M is the number of output channels, N is the number of input channels, $\mathbf{f}_{ji}$ is the i-th channel of the j-th filter, $\mathbf{x}_i$ is the i-th input channel and $\mathbf{y}_j$ is the j-th output channel. When the input data is 2D, $\mathbf{f}_{ji}$ is a 2D kernel and $\mathbf{f}_{ji} * \mathbf{x}_i$ is a 2D convolution. For this paper, we use 2D input data in all experiments. An incomplete convolution layer does the following computation instead:

$$\mathbf{y}_j = \sum_{i=1}^{N} \gamma_i (\mathbf{f}_{ji} * \mathbf{x}_i),$$

where γ_i is the profile coefficient for the i-th channel of a filter, as illustrated in Figure 3.

3) Incomplete Depthwise Separable Convolution Layer: An incomplete depthwise separable convolution [2] consists of depthwise convolution followed by pointwise convolution. To simplify the presentation in this work, IDP is only applied

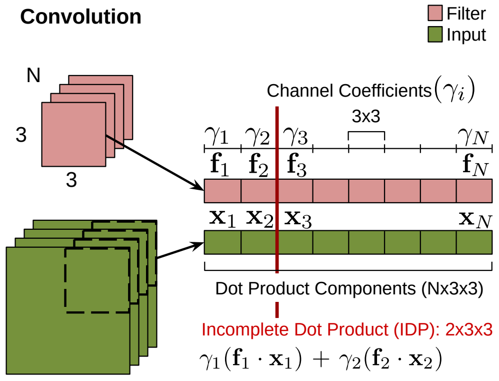
Convolution
🟥 Filter
🟩 Input
N
3
3
Channel Coefficients (γ_i)
3x3
γ_1 γ_2 γ_3 γ_N
$\mathbf{f}_1$ $\mathbf{f}_2$ $\mathbf{f}_3$ $\mathbf{f}_N$
$\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$ $\mathbf{x}_N$
Dot Product Components (Nx3x3)
Incomplete Dot Product (IDP): 2x3x3
$\gamma_1(\mathbf{f}_1 \cdot \mathbf{x}_1) + \gamma_2(\mathbf{f}_2 \cdot \mathbf{x}_2)$

Fig. 3: IDP computation for a filter $\mathbf{f}$ (green) consisting of N 3×3 kernels $\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_N$ and a patch of the input $\mathbf{x}$ (green) consisting of slices from N input channels, $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$ which is highlighted by the dashed black lines. On the right, the filter and input are shown in vector form. The coefficients $\gamma_1, \dots, \gamma_N$ correspond to each input channel. The vertical dashed line (red) signifies an IDP where only the first two channels are used to compute the dot product $\gamma_1(\mathbf{f}_1 \cdot \mathbf{x}_1) + \gamma_2(\mathbf{f}_2 \cdot \mathbf{x}_2)$.

to the pointwise convolution. A standard depthwise separable convolution layer does the following computation:

$$\mathbf{y}_j = \sum_{i=1}^{N} \mathbf{g}_{ji} * (\mathbf{f}_i * \mathbf{x}_i),$$

where $j \in \{1, \dots, M\}$, M is the number of output channels, N is the number of input channels, $\mathbf{g}_{ji}$ is the i-th channel of the j-th pointwise filter, $\mathbf{f}_i$ is the i-th channel of the depthwise filter, $\mathbf{x}_i$ is the i-th input channel and $\mathbf{y}_j$ is the j-th output channel. An incomplete depthwise convolution layer does the following computation instead:

$$\mathbf{y}_j = \sum_{i=1}^{N} \gamma_j (\mathbf{g}_{ji} * (\mathbf{f}_i * \mathbf{x}_i)),$$

where γ_j is the profile coefficient of the j-th pointwise filter, as illustrated in Figure 4.

D. Incomplete Blocks

An incomplete block consists of one or more incomplete layers, batch normalization and an activation functions. In this paper, Conv denotes an incomplete convolution block containing an incomplete convolution layer, batch normalization and an activation function. Similarly, S-Conv denotes an incomplete depthwise separable convolution block. FC denotes an incomplete fully connected block. B-Conv denotes an incomplete binary convolution block, where the layer weight values are either -1 or 1, and B-FC denotes an incomplete binary fully connected block.

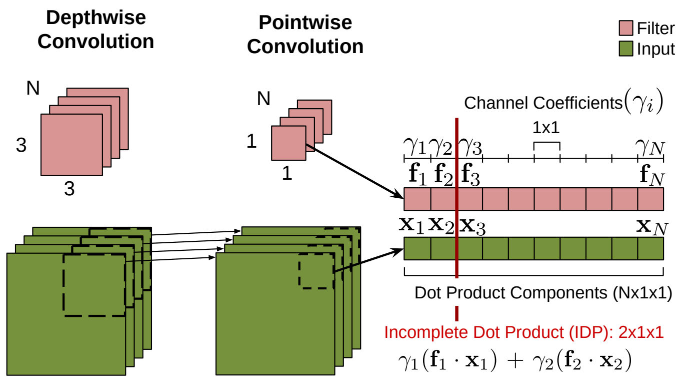
Depthwise Convolution
Pointwise Convolution
Filter
Input
N
3
3
N
1
1
Channel Coefficients(γ_i)
1x1
$\gamma_1 \gamma_2 \gamma_3$
γ_N
$\mathbf{f}_1\ \mathbf{f}_2\ \mathbf{f}_3$
$\mathbf{f}_N$
$\mathbf{x}_1\ \mathbf{x}_2\ \mathbf{x}_3$
$\mathbf{x}_N$
Dot Product Components (Nx1x1)
Incomplete Dot Product (IDP): 2x1x1
$\gamma_1(\mathbf{f}_1 \cdot \mathbf{x}_1) + \gamma_2(\mathbf{f}_2 \cdot \mathbf{x}_2)$

Fig. 4: Incomplete depthwise separable convolution consists of depthwise convolution followed by pointwise convolution. In this illustration, IDP is only applied to the pointwise convolution. Note that IDP can also be applied to depthwise convolution such as illustrated in Figure 3.

III. Multiple-Profile Incomplete Neural Networks

Multiple profiles, each focusing on a specific IDP range, can be used in a single network to efficiently cover a larger computation range. Figure 5 provides an overview of the training process for a network with two profiles (the process generalizes to three or more profiles). In the example, the first profile operates on the 0-50% IDP range and the second profile operates on the 50-100% IDP range. The training phase is performed in multiple stages, first training profile 1 and then fixing the learned weights and training profile 2. Specifically, when training profile 1, only the channels corresponding to the 0-50% IDP range are learned. After training profile 1, profile 2 is trained in a similar manner, but only learns weights in the 50-100% IDP range while still utilizing the previously learned weights of profile 1.

At inference time, a profile is selected based on the current IDP percentage requirement. The IDP percentage is chosen by the application power management policy, which dictates computation scaling based on, *e.g.*, current battery budget of the device. For the two-profile example in Figure 5, when IDP is between 0% and 50%, profile 1 is used, otherwise profile 2 is used.

While the middle IDP layers are shared across all profiles, each profile maintains a separate first and last layer. This design choice is optional. Experimentally, we found that maintaining a separate first and last layer helps improve the accuracy of each profile by adapting the profile to the subset of channels present in the IDP range. Generally, these separate layers add a small amount of memory overhead to the base model. For instance, for VGG-16 [3] model used in our evaluation in Section IV, the additional profile adds a 64, 3x3 filter convolutional layer (first layer) and a 10 neuron fully connected layer for the profile classifier (last layer). In this case, the second profile layers translate into a 3% increase in total model size over a single profile VGG-16 network.

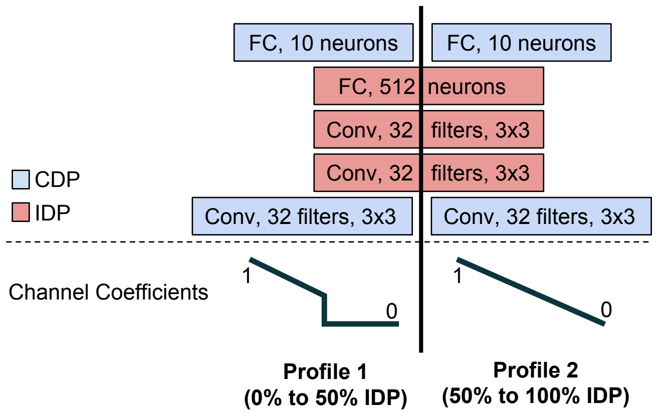

Fig. 5: Training an incomplete neural network with two profiles. First, profile 1 is trained using only the first 50% of channels in each filter for every IDP layer. Then profile 2 is trained using all channels in every IDP layer, but only updates the latter 50% of channels in each filter. Each profile has an independent first and last layer. In this example, linear profiles (see Figure 2) are used.

IV. Evaluation

In order to show the general versatility of the approach, we incorporate IDP into several well studied models and evaluate on two datasets (MNIST [4] and CIFAR-10 [5]).

A. Network Models

The network models used to evaluate IDP are shown in Figure 6. These networks cover several network types including MLPs, CNNs, Depthwise Separable CNNs (MobileNet [2]), and Binary Networks.

1) MLP: This single hidden layer MLP model highlights the effects of IDP on a simple network. We use the MNIST dataset when evaluating this network structure.

2) VGG-16: VGG-16 is a large CNN; therefore it is of interest to see how IDP performs when a large number of input channels are not used at each layer. We use the CIFAR-10 dataset when evaluating this network structure.

3) MobileNet: This network structure is interesting because of the 1x1 convolution layer over the depth dimension which is computationally expensive when depth is large, making it a natural target for IDP. We use the CIFAR-10 dataset when evaluating this network structure.

4) Binarized Networks: Binarized Networks [7] are useful in low-power and embedded device applications. We combine these networks with a device and cloud layer segmentation as shown in Figure 6. We use the MNIST dataset when evaluating this network structure.

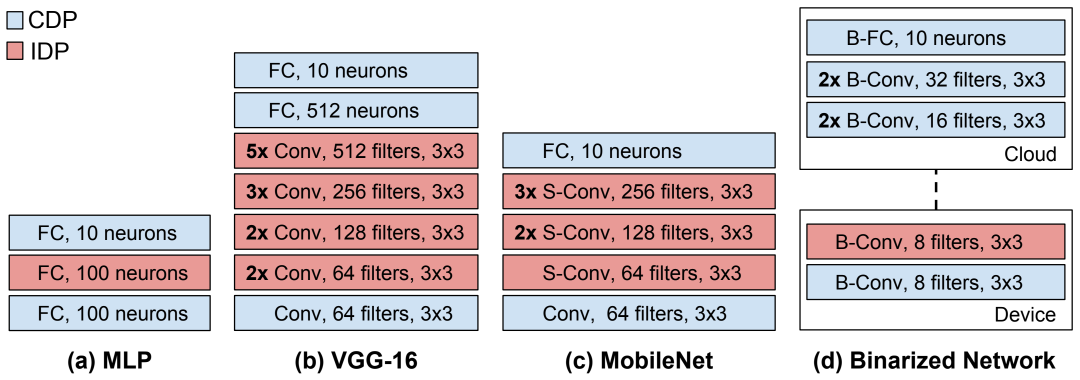

Fig. 6: The networks used to evaluate IDP. The complete dot product (CDP) layers (blue) are standard convolutional layers. The IDP layers (red) are trained with a profile, described in Section II-B, in order to use a subset of dot product components during inference across a dynamic computation scaling range. A multiplier (*e.g.*, **2x**) denotes multiple repeated layers of the same type. For MobileNet, each *S-Conv* layer is a depthwise separable convolution layer as described in [2]. For the Binarized Network model, *B-Conv* and *B-FC* refer to a binary convolution layer and a binary fully connected layer, respectively. For this model, we segment the layer between the device and the cloud, with the lower layers on device and higher layers in the cloud, as in [6]. In the evaluation, when an IDP percentage is specified, such as 50% IDP, all IDP layer shown will use this percentage of input channels during forward propagation.

B. Impact of Profile Selection

The choice of profile has a large impact on the IDP range of a trained model. Figure 7 shows the effect of different profiles for the MLP. The MLP is used for this comparison since the smaller model size magnifies the differences in behavior of networks trained with the various profiles. Using the all-one profile (shown in red) is equivalent to training a standard network without IDP. We see that this all-one model achieves the highest accuracy result when the dot product is complete (100% IDP), but falls off quickly with poorer performance as IDP is decreased. This shows that the standard model does not support a large dynamic computation range. By equally weighting the contribution of each channel, the quality of the network deteriorates more quickly as fewer of the channels are used.

The other profiles weight the early channels more and the latter channels less. This allows a network to deteriorate more gradually, as the channels with the smaller contribution are unused in the IDP. As an example, the MLP model trained with the linear profile is able to maintain higher accuracy from 60-100% IDP. At 100% IDP, the linear profile model still achieves the same accuracy as the standard model. We will compare the standard network (all-one profile) to linear profile for the remaining analysis in the paper.

Compared to the linear profile, the harmonic profile places a much higher weight on the beginning subset of channels. This translates to a larger dynamic computation range, from 30-100% IDP, for a model trained with the harmonic profile. However, this model performs about 1% worse than the standard model when all channels are used (100% IDP). Fortunately, as described in Section IV-D, we can generally use multiple profiles in a single network to mitigate this problem of lowered accuracy in high IDP percentage regions.

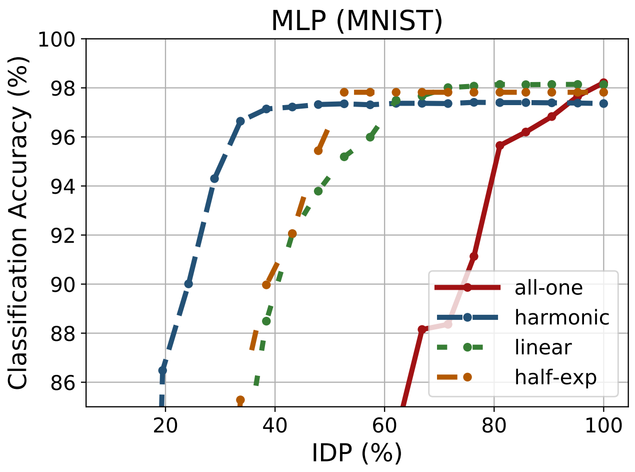

Fig. 7: A comparison of the classification accuracy of the MLP structure in Figure 6 (a), trained using the four profiles (shown in Figure 2) for the MNIST dataset. The x-axis shows IDP (%), which is the percentage of components used in the IDP layer during forward propagation. The all-one profile is equivalent to the standard network (without channel coefficients).

C. Single-Profile Incomplete Neural Networks

In this section, we evaluate the performance of the networks presented in Figure 6 trained with the linear profile compared to a standard network. Figure 8 compares the dynamic scaling of incomplete dot products for standard networks to incomplete networks over four different networks: MLP, VGG-16, MobileNet and Binarized Networks.

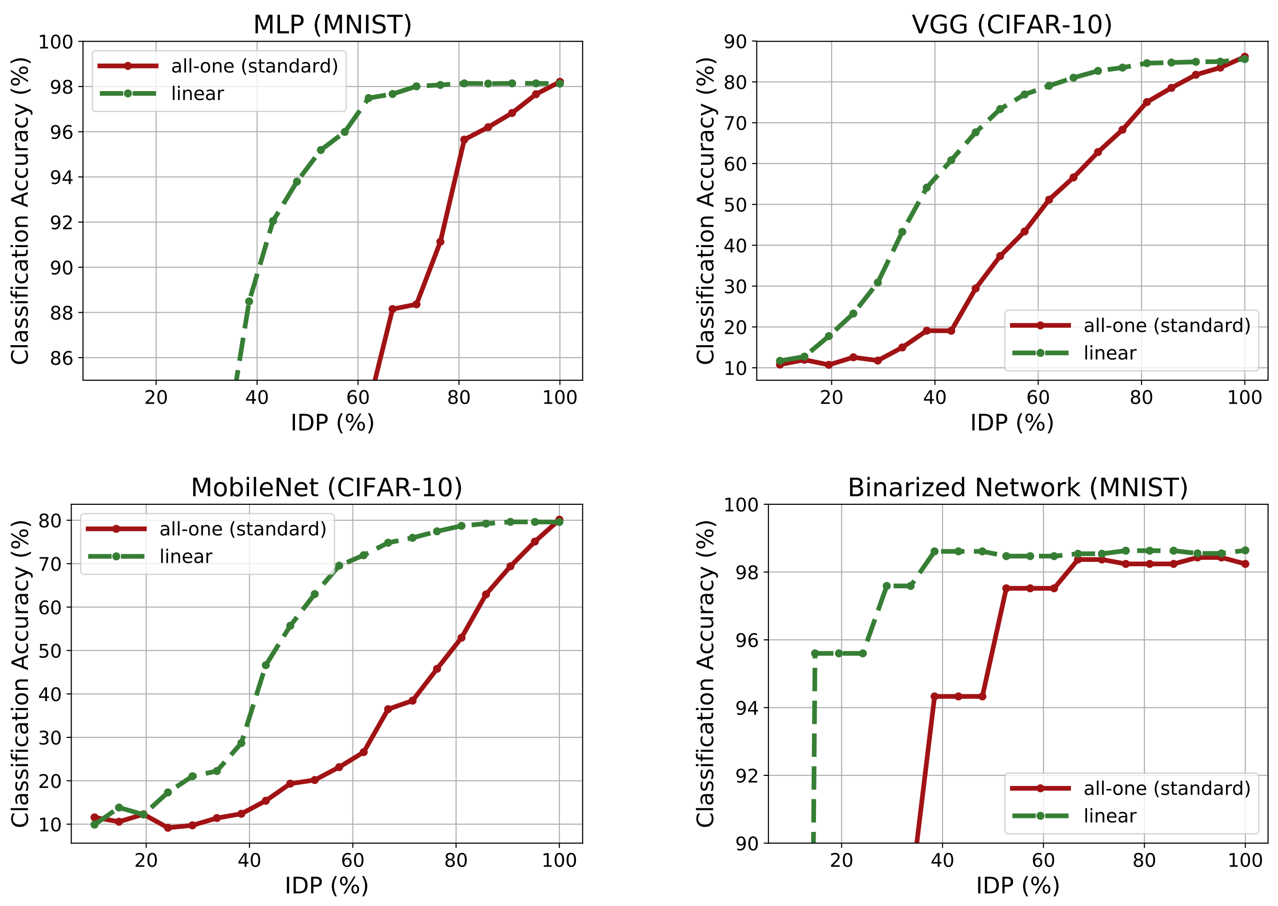

Fig. 8: A comparison of the performance of dynamic scaling with IDP for standard networks (trained without a profile) to incomplete networks (trained with the linear profile), for each network structure described in Section IV-A. Top-left is MLP. Top-right is VGG-16. Bottom-left is MobileNet and bottom-right is Binarized Network.

For each network structure, we observe that using the linear profile allows the network to achieve a larger dynamic computation range compared to the standard network. For the VGG-16 (CIFAR-10) model, at 50% IDP, the model with the linear profile achieves an accuracy of 70% as opposed to 35% by all-one (standard) model. Additionally, for each network the linear IDP network is able to achieve the same accuracy as the standard network when all channels are used.

D. Multiple-Profile Incomplete Neural Networks

In this section, we explore the performance of multiple-profile incomplete neural networks, as described in Section III. Figure 9 shows how each profile in a multiple-profile incomplete network scales across the IDP percentage range for the MLP and VGG-16 networks. For the MLP, a three profile scheme is used, where the first profile is trained for the 0-20% range, the second profile for the 20-40% range, and the third profile for the 40-100% range. For VGG-16, a two profile scheme is used, where the first profile is trained for the 0-30% range and the second profile for the 30-100% range. The profiles are trained incrementally, starting with the first profile. Each profile only learns the weights in its specified range (*e.g.*, 30-100%), but utilizes the weights learned by earlier profiles (*e.g.*, 0-30%). In this way, the later profiles can use weights learned from the earlier profiles without affecting the performance of the earlier profiles. Training the network in this multi-stage fashion enables a single set of weights to support multiple profiles.

For the first profile of the VGG-16 model, we observe that the accuracy does not improve when an IDP of greater than 30% is used. Since 30% IDP is the maximum for the profile, it does not learn the channels in the 30-100% IDP range, and therefore cannot use the higher ranges during inference. The second profile is able to achieve a higher final accuracy than the first profile but performs worse in the lower part of the IDP range. The two profiles are able to achieve a higher accuracy across the entire IDP range compared to the single profile network shown in Figure 8.

By training the profiles in a multi-stage fashion, the first profile is restricted to learn a classifier using only the first 30% of channels. This improves the accuracy of the model in

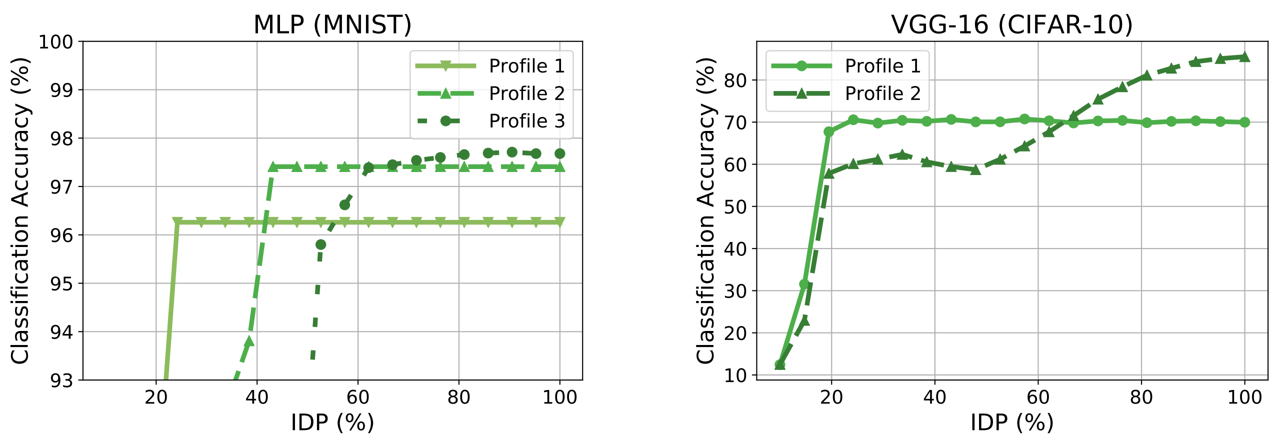

Fig. 9: A comparison of performance of dynamic IDP scaling under three profiles for MLP (left) and two profiles for VGG-16 (right). All profiles for a given model share the same network weights in the IDP layers. Both networks are trained using the linear coefficients.

the lower IDP regions compared to the single-profile case. For instance, profile 1 achieves a 70% classification accuracy at 30% IDP compared to only 30% accuracy at 30% IDP in the single profile case. While profile 2 does not update the channels learned by profile 1, it still utilizes them during training. Note that profile 2 can still achieve similar performance to the single-profile model in the 80-100% IDP region.

For the MLP model, we observe similar trends, but applied to a three profile case. As more profiles are added, we see that the final profile (profile 3) is still able to achieve a high final accuracy, even though the beginning 40% of input channels in the IDP layer are shared between all three profiles.

V. Related Work

In this section, we first compare IDP to methods that are similar in style but have the objective of preventing overfitting instead of providing a dynamic mechanism to scale over a computation range. Dropout [8] is a popular technique for dealing with overfitting by using only a subset of features in the model for any given training batch. At test time, dropout uses the full network whereas IDP allows the network to dynamically adjust its computation by using only a subset of channels. DropConnect [9] is similar to Dropout, but instead of dropping out the output channels, it drops out the network weights. This approach is similar to IDP, as IDP is also a mechanism for removing channels in order to support dynamic computation scaling. However, IDP adds a profile to directly order the contribution of the channels and does not randomly drop them during training. DeCov [10] is a more recent technique for reducing overfitting which directly penalizes redundant features in a layer during training. At a high level, this work shares a similar goal with multiple-profile IDP, by aiming to create a set of non-redundant channels that generalizes well given the restricted computation range of a single profile.

There is a growing body of work on CNN models that have a smaller memory footprint [11] and are more power efficient. One of the driving innovations is depthwise separable convolution [12], which decouples a convolutional layer into a depthwise and pointwise layer, as shown in Figure 4. This approach is a generalization of the inception module first introduced in GoogLeNet [13]–[15]. MobileNet [2] utilized depthwise separable convolutions with the objective of performing state of the art CNN inference on mobile devices. ShuffleNet [16] extends the concept of depthwise separable convolution and divides the input into groups to further improve the efficiency of the model. Structured Sparsity Learning (SSL) [17] constrains the structure of the learned filters to reduce the overall complexity of the model. IDP is orthogonal to these approaches and can be used in conjunction with them, as we show by incorporating IDP in MobileNet.

VI. Future Work

A profile targeted for a specific application can use any number of channels in the given IDP range. The current implementation approach, as described in Section III of this paper, aims to have the profiles share the same coefficients on overlapping channels. For instance, in Figure 5, the first half of the profile 2 coefficients are the profile 1 coefficients. To this end, we train the weights incrementally starting with the innermost profile and extending to the outermost profile. In the future, we want to study a more general setting, where this coefficient sharing constraint could be relaxed by jointly training both the network weights and the profile coefficients.

VII. Conclusion

This paper proposes incomplete dot product (IDP), a novel way to dynamically adjust the inference costs based on the current computation budget in conserving battery power or reducing application latency of the device. IDP enables this by introducing profiles of monotonically non-increasing channel coefficients to the layers of CNNs. This allows a single network to scale the amount of computation at inference time by adjusting the number of channels to use (IDP percentage). As illustrated in Figure 1, IDP has two sources for its computation saving, and their effects are multiplicative. For example, 50% IDP will lead to reduce computation by 75%.

Additionally, we can improve the flexibility and effectiveness of IDP at inference time by introducing multiple profiles. Each profile is trained to target a specific IDP range. At inference time, the current profile is chosen by the target IDP range, which is selected by the application or according to a power management policy. By using multiple profiles, we are able to train a network which can run over a wide computation scaling range while maintaining a high accuracy (see Figure 9).

To the best of our knowledge, the dynamic adaptation approach of IDP as well as the notion of multiple profiles and network training for these profiles are novel. As CNNs are increasingly running on devices ranging from smartphones to IoT devices, we believe methods that provide dynamic scaling such as IDP become more important. We hope that this paper can inspire further work in dynamic neural network reconfiguration, including new IDP designs, training and associated methodologies.

VIII. Acknowledgements

This work is supported in part by gifts from the Intel Corporation and in part by the Naval Supply Systems Command award under the Naval Postgraduate School Agreement No. N00244-16-1-0018.

References

[1] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Federated learning of deep networks using model averaging," *CoRR*, vol. abs/1602.05629, 2016. [Online]. Available: http://arxiv.org/abs/1602.05629

[2] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

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

[4] Y. LeCun, C. Cortes, and C. J. Burges, "The mnist database of handwritten digits," 1998.

[5] A. Krizhevsky, V. Nair, and G. Hinton, "The cifar-10 dataset," 2014.

[6] S. Teerapittayanon, B. McDanel, and H. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *37th International Conference on Distributed Computing Systems (ICDCS 2017)*, 2017.

[7] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[8] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[9] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 2013, pp. 1058–1066.

[10] M. Cogswell, F. Ahmed, R. Girshick, L. Zitnick, and D. Batra, "Reducing overfitting in deep networks by decorrelating representations," *arXiv preprint arXiv:1511.06068*, 2015.

[11] B. McDanel, S. Teerapittayanon, and H. Kung, "Embedded binarized neural networks," *arXiv preprint arXiv:1709.02260*, 2017.

[12] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," *arXiv preprint arXiv:1610.02357*, 2016.

[13] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[14] 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.

[15] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning." in *AAAI*, 2017, pp. 4278–4284.

[16] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," *arXiv preprint arXiv:1707.01083*, 2017.

[17] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 2074–2082.